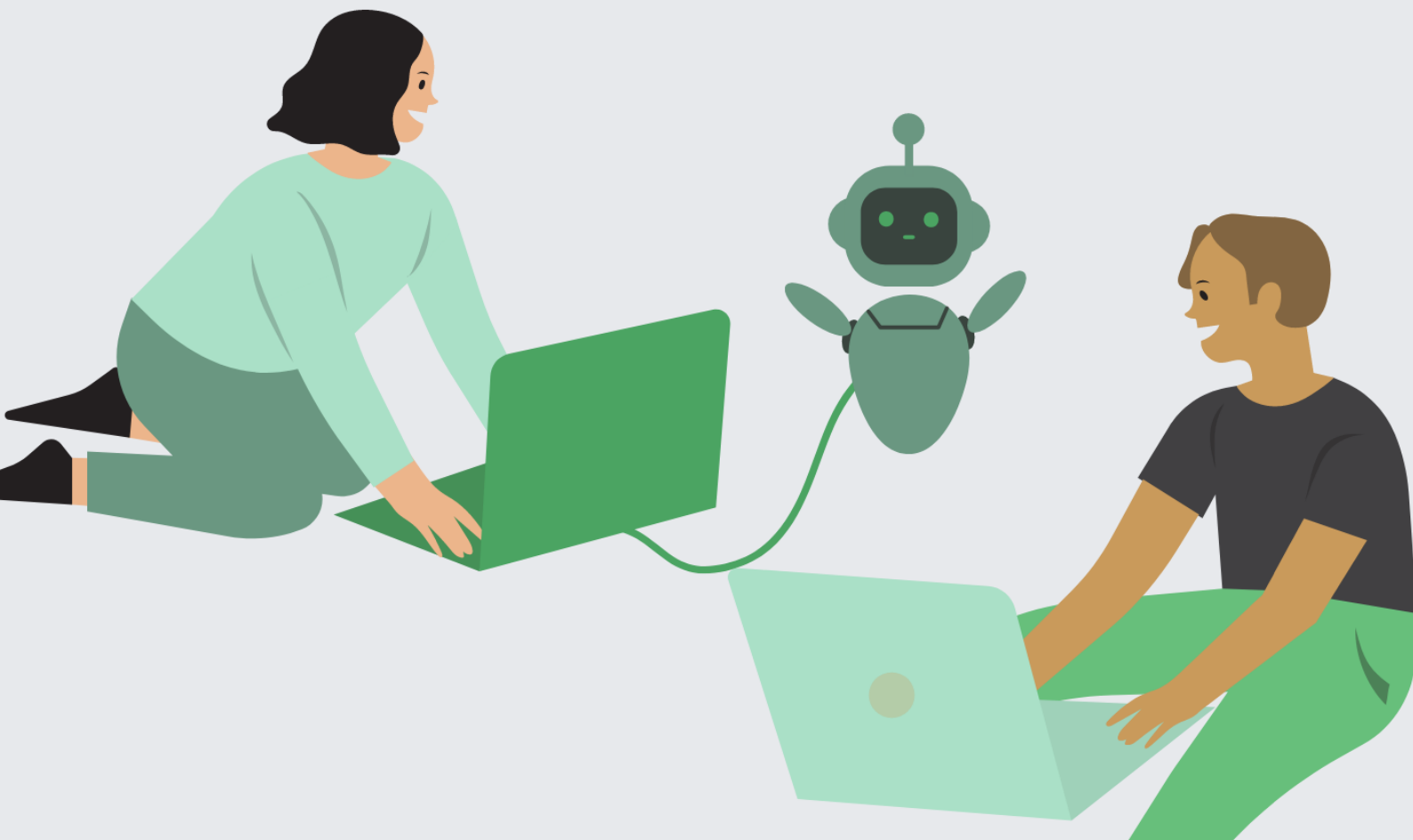




The CodER Module on Coding & Microcontrollers: 20 hours lessons



Contents

Introduction	4
1. Context of the CodER Project.....	5
1.1. Looking at the bigger picture: Gamification and its role in education	5
1.2. Aim of game-based education methods in coding and microcontrollers.....	6
1.3. The audience and target group of these activities	8
PART A: The CodER Coding Module (10 hours).....	11
1. Introduction to programming	12
1.1. What is programming	12
1.2. Why learn programming? What are the benefits?.....	12
1.3. The process of program development and algorithms.....	13
1.4. Programming languages – The Most In-Demand Programming Languages.....	15
2. Getting Setup with Python.....	16
2.1. What is Python?.....	16
2.2. The most used Python Integrated Development Environments (IDEs) per industry	17
2.3. Setting up a Python Integrated Development Environment (IDE)	18
2.4. Running Python in Command line	23
3. Basics to programming in Python.....	24
3.1. The Basics: Data Types (Basic and Complex).....	24
3.2. Variables, expressions, and statements.....	25
3.3. Lists, Dictionaries and Tuples	30
3.3.1. Lists.....	30
3.3.2. Tuples	35
3.3.3. Dictionaries.....	39
3.4. Conditionals and Loops	43
3.4.1. Boolean expressions.....	43
3.4.2. Logical operators.....	45
3.4.3. Nested if statements	46
3.4.4. Loops	46
3.5. Understanding the flow of execution through functions.....	49
3.6. Putting the pieces together - How to build a program.....	53
3.7. How to adjust a program to fit your needs	56
3.8. Syntactic, Runtime and Semantic Errors – Handling Errors in Python.....	57



3.9. Practice.....	59
PART B: The CodER Microcontrollers Module (10 hours).....	60
1. Introduction to Microcontrollers.....	61
1.1. What is a microcontroller.....	61
1.2. What is Arduino and its different types.....	62
1.3. Concepts: Input, Output, Analog, Digital.....	66
2. Fundamentals of Programming with Arduino IDE.....	69
2.1. Setting up Arduino IDE and basic commands.....	69
2.2. Programming in Arduino and uploading programs to the board.....	72
2.3. Blinking Led with Arduino.....	74
3. Applications.....	77
3.1. What is robotics?.....	77
3.2 Types of robots.....	77
3.3 Driving a DC motor with a motor shield.....	82
3.4. Build a paintbot using a DC motor and Arduino.....	84
3.5 Build an interactive paper toy with a servo motor.....	87
3.6. Remote-controlled door lock.....	94
3.7. Measuring temperature, humidity, light and colour.....	99
3.7.1. Using a sensor with Arduino.....	100
3.7.2. Build a theremin with Arduino and a light sensor.....	101
3.7.3. Colour sensitive robots.....	102
3.7.4. Introduction to DHT11 sensor.....	103
3.7.5. Build a smart cooling fan.....	106



Introduction

In the modern world, there is an increasing emphasis on the acquisition of digital skills. However, the current mismatch of supply and demand found in the labour market presents new challenges for employers and jobseekers alike. This mismatch primarily affects young people since youth unemployment is one of the most significant issues that Europe is facing. The CodER project represents an attempt to bridge this gap by materialising knowledge on coding and microcontrollers through the innovative method of Escape Rooms (ERs) for youth workers and organisations to educate young people and increase their interest and engagement towards occupations that are tech-oriented. The European partners that are taking part in this project, funded by the Erasmus+ program, are: Digijeunes (France), Challedu (Greece), Citizens in Power – C.I.P. (Cyprus), RITE (Cyprus), AKMI (Greece), and Kalimera (Croatia).

Through this project, youth workers and organisations will be equipped with new skills and innovative methods to attract young people, and especially young women, in tech-oriented pathways. The CodER Module represents the first step towards the realisation of this project's aims for the general public. This module provides the basics of programming and microcontrollers based on real-life examples to make its content relatable to everyday usage. It attempts to instil the logic behind programming and microcontrollers to promote the cultivation of critical thinking, creativity, and problem-solving skills, amongst other skills.

The context and approach of the CodER project are presented initially to provide the readers with an understanding of the rationale behind the gamified/game-learning approach taken. The first part of the module is dedicated to understanding the use and benefits of programming, setting up Python Integrated Development Environments (IDEs), and going through the basics of the Python programming language to build a small program. The second part of the module is dedicated to introducing Microcontrollers and their usage, setting up the Arduino software and learning how to use it to execute small tasks on microcontrollers.



1. Context of the CodER Project

1.1. Looking at the bigger picture: Gamification and its role in education

Over the past decade, there has been a shift from traditional teacher-centric learning methods to what is sometimes referred to as learner-centered approaches to education. Whereas traditional approaches to education involve "the passive transfer of knowledge from the instructor to the student" (McGuire & Gubbins, 2010, p.1), in learner-centered approaches to education, "students are expected to become aware of and evaluate their own experience (...) where the instructor is no longer an oracle, but a guide who participates in learning" (McGuire & Gubbins, 2010, p.1). This contemporary approach to learning methods is usually more technologically-based and involves the use of interactive strategies that aim to motivate students to actively engage and participate in their learning process.

One reason for adopting these technology-based approaches to education, and especially to STEM education, is that a significant number of jobs are now centered around activities and tasks that require the use of internet and digital technologies (Daniel Calderón-Gómez et al. 2020), and in the near future even more jobs will have to adopt this structure. This means that these jobs will require the acquisition of at least some basic digital skills, whereas more favourable and high-paid jobs will require an even more advanced knowledge of digital technologies (Karpinski et al., 2021). Another factor that played a role in moving towards learner-centered approaches to education, except the rapid technological advancement, is the correlation found between disengagement, low performance, and low participation rates in diverse educational settings (as cited in Levels et al., 2022; Callanan et al., 2009). In the case of youth, which ranges from 15- to 29-year-olds, this is further highlighted by the statistics available in Europe where 13.7% were neither in Education, Employment or Training (NEETs) (Eurofound, 2022). Therefore, policymakers and youth organisations found themselves faced with new challenges to reinterest and reintegrate youth in education, employment and/or training.

In various levels of education, the tendency to approach learning through game-based elements in formal, non-formal and informal education contexts from early childhood to higher education and beyond has become increasingly popular. Within this context, the definitions of 'gamification' and 'game-based learning' emerged. Although the two terms are often used interchangeably, they have some slight differences. By the term 'gamification', we usually refer to "the use of game design elements in nongame contexts" (as cited in Dichev & Dicheva, 2017, p. 2; Deterding, et al., 2011; Hamari et al., 2014; Werbach, 2014). In the context of education, gamification refers to the use of "game design elements and gameful experiences in the design of learning processes" (Dichev & Dicheva, 2017, p.2). Game design elements consist of the dynamics, mechanics, and components of a game (Dichev & Dicheva, 2017). On the other hand, game-based learning (GBL) utilises the techniques deployed by game designers to create an entertaining and immersive gaming



experience for the user, with the sole aim of designing a virtual learning environment that engages its users in educational activities. This type of learning can occur both physically and digitally. Game-based learning is considered to have clearly defined learning outcomes achieved through game contents (Sanchez, 2019). It is also commonly referred to as serious games, digital learning, or educational games (Sanchez, 2019).

There are a plethora of studies available that explore the effects of game-based learning in individuals of different ages from school students to higher education students and less frequently in adults (e.g., Dichev & Dicheva, 2017; Ninaus et al., 2017; Sanchez et al., 2020). The basis for this movement towards game-based learning is found in its potential to motivate learners, especially within the youth age range, to actively engage and participate in their own education. As described by Dichev & Dicheva (2017), such approaches to learning call for "immersion in a way similar to what happens in games" (Dichev & Dicheva, 2017, p.2). Since video games are primarily designed for entertainment, "they can produce states of desirable experience and motivate users to remain engaged in an activity" (Dichev & Dicheva, 2017, p.12). Motivation is shown to be one of the most significant factors for learners to engage and successfully perform any activity that requires time and effort. Through game-based learning, the concept of time and effort is transformed into an educational and entertaining environment that simultaneously allows users to build on their knowledge, skills, and attitudes.

The simple act of framing an activity or task as a game suffices to bring about positive psychological effects such as engagement or enjoyment (Dichev & Dicheva, 2017). This view seems to be based on the assumption that intrinsic motivation derives from the basic psychological need of satisfying that "feeling of competence or self-efficacy" (Ninaus et al., 2017, p.16) present in solving a problem or in our context, completing a game objective. The capacity of games to draw the players' attention into a state of absorption while engaging them in problem-solving tasks, stems from their goal-driven nature that places the player in a state of absorption in trying to complete game objectives. There is a pedagogical dimension inherent in games that is usually ignored, as Tulloch (2014) suggests, namely "the process of games teaching players how to play" (as cited in Dichev & Dicheva, 2017, p.23). Therefore, game-based methods of learning seek to imitate how a game draws the players in a state of complete absorption and engagement in order to create an equally immersive experience for educational purposes.

1.2. Aim of game-based education methods in coding and microcontrollers

Since the pandemic started, the emphasis placed on integrating technology into different industries became even more prominent. The vision of the European Commission presented in 2021 encompasses four vital points to enable Europe's digital transformation by 2030. One of these critical points is the digital competence needed to ensure the development of



a resilient European economy and society. The target set to be reached by 2030 is for 80% of people to have basic digital skills (European Commission, 2021a).

However, the biggest obstacle to the realisation of a European digital transformation is the lack of digital skills in relation to their high demand. The percentage of employers facing a shortage of digitally competent employees is as high as 70% (European Commission, 2021b). This enlarges the digital skills gap that exists in the labour market, where even low-skilled to semi-skilled professions require a level of digital competence (Karpinski et al., 2021). As stated in the Digital Economy and Society Index (DESI), 56% of Europeans possess basic digital skills, but only 31% possess digital skills above the basic level (European Commission, 2021b). Thus, the importance of problem-solving skills and computational thinking is high on the agenda since only 13% of young people and 6% of adults possess such skills in the EU (Eurostat, 2020).

The required digital skills are translated into the "Digital Competence Framework for Citizens" (DigComp) developed by the European Commission, which incorporates a variety of skills: 1) Information and data literacy, 2) Communication and collaboration, 3) Digital content creation, 4) Safety and 5) Problem-solving (Centeno et al., 2019). It is argued that the new technological requirements of occupations will create a new division within the labour market, represented by three categories of digital workers: "the 'digital precariat' (with a poor economic situation); 'traditional digital labour' (mainly involved in productive digital tasks); and the 'innovative class' (carrying out productive and communicative digital tasks)" (Calderón-Gómez et al., 2020, p. 7-8). Furthermore, it has been suggested that the 'digital precariat' will predominantly consist of young people and women (Calderón-Gómez et al., 2020, p.9). In this vein, youth that is already at risk of exclusion in society and the labour market based on their inactivity or inability to find employment or pursue further education and training are found at the crossroads of double exclusion.

Corneliussen (2021) also highlights that the lack of female role models in these fields and the dominant narrative of male ICT professionals actively discourages young girls and women from a STEM-oriented path. Overall, more favourable job positions are generally more occupied by men than by women (Calderón-Gómez et al., 2020) As it has been mentioned above, women along with young people are the most prominent groups within the digital precariat. At the same time, women are "underrepresented in the innovative class, while also being overrepresented in traditional analogical labour" (Calderón-Gómez et al., 2020, p. 8). A study shows that it is more frequent for men to spend more time on the Internet for work-related activities than women, and between "those holding a university degree, 77.5% of men compared with 70.0% of women use the internet at work" (Calderón-Gómez et al., 2020, p.18-20). Furthermore, women tend to use digital technologies more for communication and mobile use, whereas men tend to use digital technologies more for productive and office use, and for multiple functions and uses (Calderón-Gómez et al. 2020).



In general, compared to men, women are less likely to use the internet and/or digital technologies for more complex uses.

Therefore, coding and microcontrollers are highly relevant to the current needs of the labour market. The combination of game-based learning in the realm of coding and microcontrollers allows us to reach our target group. Based on the benefits discussed in the previous section, game-based learning works as a means to transfer knowledge and skills through an entertaining and meaningful learning experience. More specifically, studies that have used Escape rooms as their learning tool have demonstrated their capacity to stimulate positive emotions, increase the level of engagement and generate an overall positive learning experience for learners (Llerena-Izquierdo & Sherry, 2022; Sánchez-Martín et al., 2020). The encouraging results of such studies present opportunities for the CodER project to be able to bridge the digital skills gaps and offer new opportunities to youth that is at risk of exclusion.

Another point that is highly relevant to our efforts to create digital escape rooms is the limitation of physical escape rooms. As Fotaris and Mastoras (2019, p.3) point out that "it is not feasible or legal to lock a subset of a class in a room and wait until they puzzle their way out (...), many escape rooms designed for the classroom have been stripped back to a group table-top activity involving a series of locked boxes". But this solution lacks the immersive character that is typical of escape rooms. There are also other challenges that need to be resolved when implementing escape rooms for educational purposes, including time commitment due to the labour-intensive process required to create escape rooms, time constraints that limit playtesting which may lead to even more problems such as poor design and unbalanced difficulty, and lastly, large group sizes which complicate the way in which students are going to participate in the escape room or how engaging it will be (Fotaris & Mastoras, 2019). In this view, the digital escape room generator that will be created as the end-result of this project resolves the majority of these constraints and allows youth workers greater flexibility to design such spaces to match their learners' needs.

Moreover, Llerena-Izquierdo & Sherry (2022) highlight that the lack of design guides and experience from users are two limiting factors for the wide adaptation of such tools in formal, non-formal, and informal educational contexts. This is another important point that we attempt to address with the creation of this module as a first step, and the other expected project results that will equip youth workers with the necessary knowledge and skills to instil new knowledge to youth.

1.3. The audience and target group of these activities

The audience that the CodER project is addressing is members of local, national, European youth organisations. This includes all types of professionals working for a youth organisation, dealing with disadvantaged groups such as NEET, early school leavers, migrants, refugees and asylum seekers, and ex-prisoners. Also, youth organisations that are



focusing on the provision of non-formal education related to technology and innovation, youth organisations and/or youth centres that are seeking to increase youth employability and youth organizations dealing with STEM education are also part of the target audience along with youth workers dealing with programming, microcontrollers, educational escape rooms. Additionally, the target audience of CodER extends to IT experts, representatives of ICT start-ups, scientists, researchers, university staff, opinion makers, and influential multipliers such as coding associations, innovation centres, representatives of state bodies, small and medium-sized enterprises, social and sectoral bodies, product marketing experts and local municipal employees.

Through this audience, we aim to reach our target group, which is young people, to instil the knowledge gained by youth organisations to them. The target group includes young people who belong to displaced populations (migrants, asylum seekers, refugees, minority populations), young people who are generally at risk of socio-economic exclusion (early school leavers, NEET, etc.) and young people with learning disorders. As it was mentioned earlier, a large number of young people are struggling with unemployment due to a lack of coding skills that employers are looking for in their employees. Therefore, this project is designed in a way that will teach the basics of programming and microcontrollers to youth organisations, provide them with a methodological and pedagogical guide on how to use escape rooms to educate young people through a playful and entertaining experience, create a series of different scenarios for them to use as templates or adjust them, and create a digital escape room generator.

References

Calderón-Gómez, D., Casas-Mas, B., Urraco-Solanilla, M., & Revilla, J. C. (2020). The labour digital divide: digital dimensions of labour market segmentation. *Work Organisation, Labour & Globalisation*, 14(2), 7-30.

Centeno, C., Vuorikari, R., Punie, Y., O'Á, W., Kluzer, S., Vitorica, A., ... & Bartolome, J. (2019). *Developing digital competence for employability: Engaging and supporting stakeholders with the use of DigComp* (No. JRC118711). Joint Research Centre.

Corneliussen, H. G. (2021). Women empowering themselves to fit into ICT. In *Technology and Women's Empowerment*. Taylor & Francis

Dichev, C. & Dicheva, D. (2017). Gamifying education: what is known, what is believed and what remains uncertain: a critical review. *International Journal of Educational Technology in Higher Education* 14, 9. <https://doi.org/10.1186/s41239-017-0042-5>

DigitalEurope (n.d.). Key indicators for a stronger digital Europe. <https://www.digitaleurope.org/key-indicators-for-a-stronger-digital-europe/>

European Commission (2021a). *Europe's Digital Decade: digital targets for 2030*. https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/europes-digital-decade-digital-targets-2030_en



European Commission (2021b). *Digital skills and jobs*. <https://digital-strategy.ec.europa.eu/en/policies/digital-skills-and-jobs>

Eurofound (2022). NEETs. <https://www.eurofound.europa.eu/topic/neets>

Eurostat (2020). *Being young in Europe today – digital world*. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Being_young_in_Europe_today_-_digital_world&oldid=528990#Information_and_communications_technology_skills

Fotaris, P., & Mastoras, T. (2019). Escape rooms for learning: A systematic review. In *Proceedings of the European Conference on Games Based Learning*, 235-243.

Karpinski, Z., Biagi, F., & Di Pietro, G. (2021). Computational Thinking, Socioeconomic Gaps, and Policy Implications. IEA Compass: Briefs in Education. 12. *International Association for the Evaluation of Educational Achievement*

Levels, M., Brzinsky-Fay, C., Holmes, C., Jongbloed, J., & Taki, H. (2022). The dynamics of marginalized youth: *Not in education, employment, or training around the world*. Taylor & Francis.

Liu, Z.-Y., Shaikh, Z. A., & Gazizova, F. (2020). Using the Concept of Game-Based Learning in Education. *International Journal of Emerging Technologies in Learning (IJET)*, 15 (14), 53–64. <https://doi.org/10.3991/ijet.v15i14.14675>

Llerena-Izquierdo, J., & Sherry, L. L. (2022). Combining Escape Rooms and Google Forms to Reinforce Python Programming Learning. In Á. Rocha, P. C. López-López & J. P. Salgado-Guerrero (Eds.), *Communication, Smart Technologies and Innovation for Society* (pp. 107-116). Springer, Singapore.

Mcguire, D. & Gubbins, C. (2010). The Slow Death of Formal Learning: A Polemic. *Human Resource Development Review*. 9. 249-265. 10.1177/1534484310371444.

Ninaus, M., Moeller, K., McMullen, J., & Kiili, K. (2017). Acceptance of Game-Based Learning and Intrinsic Motivation as Predictors for Learning Success and Flow Experience. *International Journal of Serious Games*, 4(3), 15–30.

Sanchez, E. (2019) Game-Based Learning. In: Tatnall A. (eds) *Encyclopedia of Education and Information Technologies*. Springer, Cham. https://doi.org/10.1007/978-3-319-60013-0_39-1

Sánchez-Martín, J., Corrales-Serrano, M., Luque-Sendra, A., & Zamora-Polo, F. (2020). Exit for success. Gamifying science and technology for university students using escape-room. A preliminary approach. *Heliyon*, 6(7). <https://doi.org/10.1016/j.heliyon.2020.e04340>

Vasilescu, M. D., Serban, A. C., Dimian, G. C., Aceleanu, M. I., & Picatoste, X. (2020). Digital divide, skills and perceptions on digitalisation in the European Union—Towards a smart labour market. *PLoS ONE*, 15(4), 1–39. <https://doi.org/10.1371/journal.pone.0232032>



PART A: The CodER Coding Module (10 hours)

Description:

In this part, a comprehensive introduction to programming is provided along with its usage and application based on real-life examples. The benefits of programming and its application to the job market are initially explained, as well as the process followed by an algorithm and program development. Then, the second subsection explains what Python is, the usage of an Integrated Development Environment (IDE) and how to set it up. The rest of the module is dedicated to the different data types, their usage and application through examples, as well as the development of short programs.

Workload:

10 hours

Learning outcomes:

By the end of this course, learners will be able to:

- ⇒ Recognise the value and usage of programming
- ⇒ Comprehend the flow of execution in programs
- ⇒ Use basic syntax to access, modify, and delete different data types in Python
- ⇒ Use Python to build small programs

Required material and resources:

- ⇒ Computer or laptop
- ⇒ Internet Access
- ⇒ Spyder
- ⇒ Visual Studio Code

Practicalities:

This part of the module is designed to cover 10 hours of learning. Each section of the module has a designated time, but the learner or educator/trainer is free to decide how much to spend on each subtopic depending on prior knowledge and engagement in similar topics. The content is based on progressive development and is used to provide basic knowledge and skills to beginners.

Subsections:

1. Introduction to Programming
2. Getting Setup with Python
3. Basics to programming in Python



1. Introduction to programming

- ⇒ **Number of participants:** 1-10 per facilitator
- ⇒ **Duration:** 0.5-0.75h
- ⇒ **Teaching methods:** Lecture, presentation
- ⇒ **Required materials:** Presentation

1.1. What is programming

Programming is the process of creating a program dedicated to completing a specific task through a computer.

Programs include a step-by-step procedure, usually described as creating a recipe, that is used to communicate with the computer by using a series of predefined syntax and functions.

You might think that this sounds overly complicated, but it is not. It is like learning a new natural language from scratch, you have to understand its syntax and vocabulary to be able to use it properly. An equally important aspect is practising your skills to get better at any natural or programming language that you are learning.

Natural language (English)	Programming language
John reads every day	print("Hello, World")

Table 1 - Comparison between natural and programming language

1.2. Why learn programming? What are the benefits?

You might be wondering what the purpose of learning a programming language is and the benefits that it entails.

According to Steve Jobs: "Everyone in this country should learn to program a computer because it teaches you to think". Based on this quote, one of the major benefits of programming is that it develops computational thinking. Computational thinking "refers to one's ability to identify, test, and implement possible algorithmic solutions to the problem at hand and to analogous problems that might arise in a new context or situation" (Karpinski et al., 2021, p. 3). This also directly relates to problem-solving skills, which are considered as desirable and fundamental skills to have to be able to adapt to the changes and demands of the current labour market (as cited in Karpinski et al., 2021; Czaja and Urbaniec, 2019; Rakowska and Cichorzewska, 2016; Slavinskis et al., 2015). Problem-solving skills are held in high esteem amongst employers, and they are considered to be one of the most important soft skills of the 21st-century.



The ability to use a programming language does not signify that everyone needs to become a programmer, but that programming can be beneficial to any career and can open up new career pathways.

The flexibility that programming allows is another one of its major assets since new job opportunities will emerge that are more code-oriented. In addition to this, basic knowledge of programming will be necessary for any job in the future. This increases its value as a fundamental skill to tomorrow's society and labour market.

Any person that is feeling stagnant in their career can learn a programming language to increase their income and career prospects, as well as take their problem-solving skills a step further. Therefore, programming is an all-encompassing skill that requires both technical and soft skills and can be learnt by anyone that is willing to try.

1.3. The process of program development and algorithms

If you are still reading this module, then let us explain what an algorithm is and how a program is developed.

Algorithms are an integral part of programs since they represent the steps that are required to complete the task specified by the code. A good algorithm must include the following: the steps or activities that are required to complete the task, the correct order of those activities and their termination. A frequent example that is used to understand the process of an algorithm is a recipe. The following recipe of "Bake a Cake" can be translated into an algorithm:



Figure 1 - Visual representation of an algorithm

As you can see the order of every activity or step is based on a logical order. For instance, you cannot place the pan in the oven without first putting the ingredients into the pan; not if you want to have a fruitful procedure. The same applies to algorithms, every step or activity has a purpose to serve, and they should have a proper order for the algorithm to

work as we want it. The description above is also called *pseudocode*, which is used to explain the process followed by an algorithm.

Another important aspect is the pattern followed to design a specific algorithm, which usually involves the Input, Process and Output (IPO) Pattern. The algorithm begins by reading data, moves on to data manipulation, and ends by displaying a result. For example, suppose that you want to calculate the course grades of your students: you would first go through their scores on every test (input), then calculate their average grade (process), and finally, show their average grade (output). This is a very useful pattern to consider when creating an algorithm.

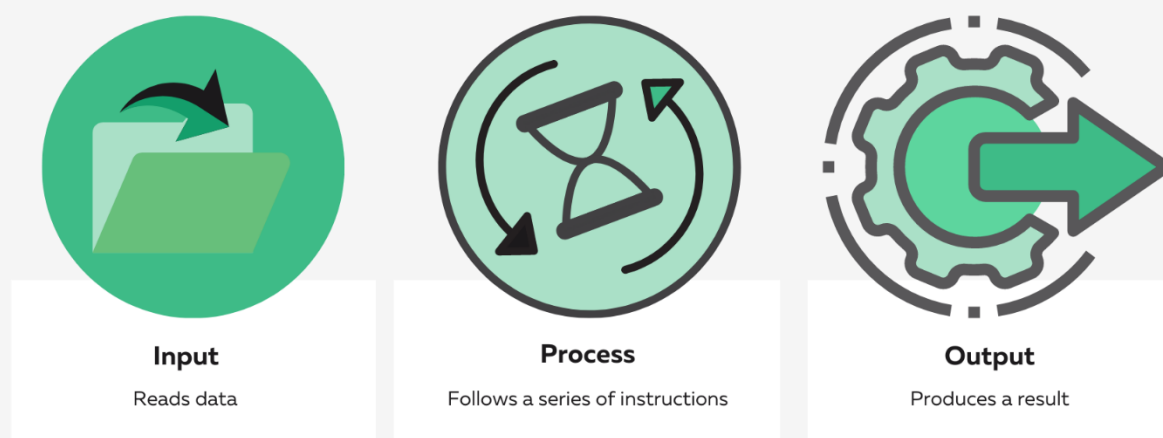


Figure 2 – IPO Pattern

Since we have learnt what an algorithm is and the process required to create a good algorithm, we can now understand a bit better how to develop a program. Program development goes through a cycle that includes analysis, design, coding, debugging and testing, and implementing and maintaining the proposed solution.

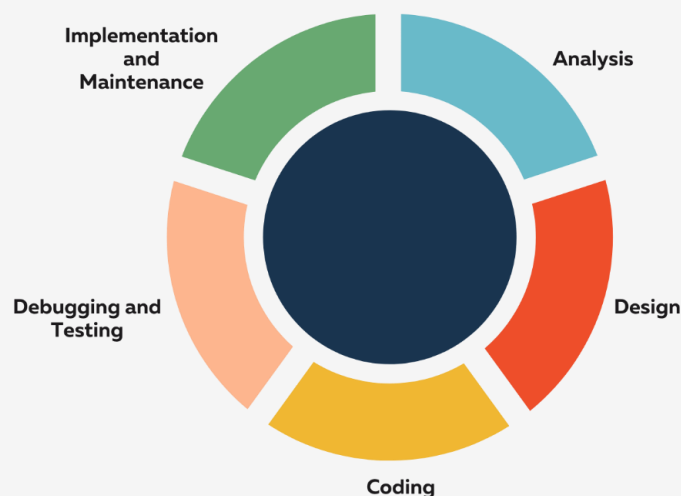


Figure 3 – Program Development Cycle

In the first step, we are trying to understand, identify and analyse the problem that requires a computational solution. The second step is all about designing the program, which entails creating a visual diagram of the process that the program will follow. This is particularly useful to help you break the problem into smaller parts. The next step is coding the program based on the visual diagram that we created in the previous step. The computer user will run the code for this step to spot any problems. In the fourth step, the user must start debugging the program in order to avoid potential errors. The fifth step requires the user to run the program and make sure that there are no syntactical or logical errors. The final and sixth step revolves around documentation and maintenance of the program, which requires an explanation of the program's rationale and its processes.

If there are some things that you have not understood from this process, do not worry. Once we start practicing, it will become clearer. In this module, we will focus on the first three steps. Therefore, we will try to explain the process of analysing a problem and developing a program that can serve as a potential solution.

1.4. Programming languages – The Most In-Demand Programming Languages

According to an analysis of thousands of job openings in the US and Europe, Python has been ranked as the most in-demand coding language for 2022, followed by Java and JavaScript. While results suggested that demand for C, C++, and C# was not as strong, it was still existent for selected professions. While Python has been around for decades, its demand in 2022 will continue to grow thanks to its exponential use in the thriving industries of data science, machine learning, and AI.

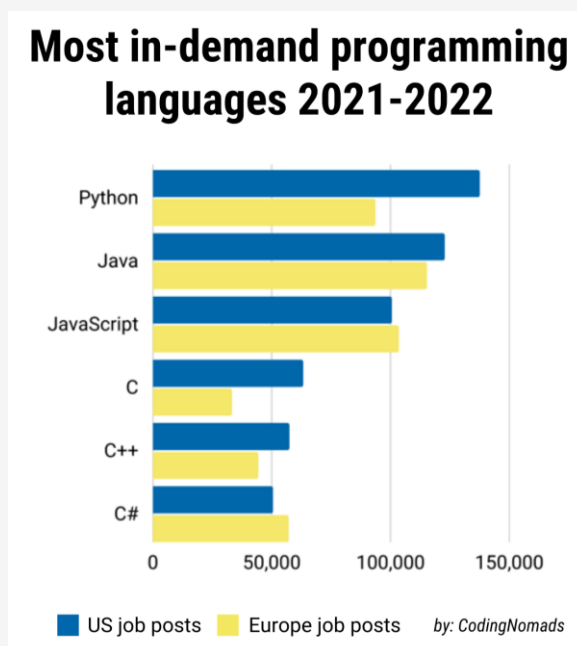


Figure 1 - Most In-Demand Programming Languages

Source: <https://www.siliconrepublic.com/careers/python-most-in-demand-coding-language-2022>



2. Getting Setup with Python

- ⇒ **Number of participants:** 1-10 per facilitator
- ⇒ **Duration:** 0.75-1h
- ⇒ **Teaching methods:** Presentation, Guided instruction, Experiential learning
- ⇒ **Required materials:** Presentation, Computers (1 per participant) and stable internet connection

Before we explain how to set up a Python Integrated Development Environment (IDE) and how to run Python in a command line, we will briefly explain what Python is and why it is the most used programming language in the labour market.

2.1. What is Python?

Python is a high-level programming language that allows greater flexibility and readability when coding and can be easily adapted on different kinds of computers with none or a few potential modifications. Other high-level languages include C, C++, and Java.

The difference between high-level and low-level programming languages is that computers execute programs written in low-level languages. Low-level languages use binary coding (0,1) to execute a program. However, that is not easy to interpret for a human unless they have the expertise. Therefore, programs that are written in high-level languages have to be transformed or translated into a binary form for the computer to understand what to do. This requires some extra processing time, but it is a relatively small disadvantage of high-level languages.

Generally, there are two ways that a program is translated into binary form to be executed. The first one involves using a compiler where the program is read and translated before it runs. As illustrated in the figure below, the process starts from the source code (for a high-level program) moves on to the object code or executor (which translates program into binary form) and goes back to compile the program to be executed as output without further translation.

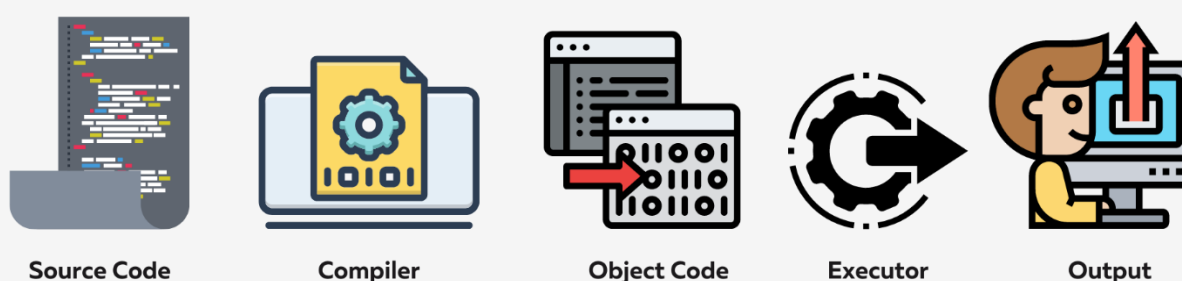


Figure 5 - Compiler language (Adapted from Downey et al., 2002, p.30)

The second way uses only an interpreter as mediator between the source code and the output. The interpreter essentially reads the program and follows its instructions line by

line. This is an ongoing back and forth process until the program is finished, as you can see in the figure below.

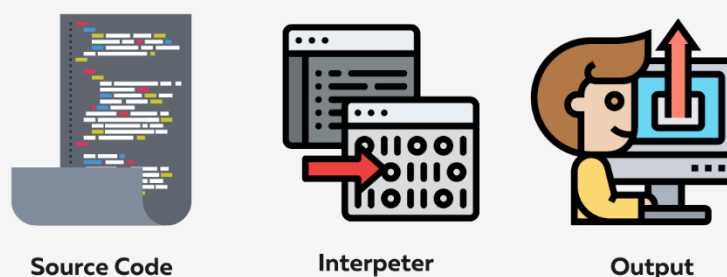


Figure 6 - Interpreter language (Adapted from Downey et al., 2002, p.30)

Python uses the interpreter to execute its programs and thus, it is considered an interpreted language. This can either be done by the command-line mode or the script mode. In the command-line mode, you write your code, and the interpreter prints the end-result. On the other hand, the script mode uses a program that contains a file ending in `.py` and uses the interpreter to execute the contents of the file. It should be noted that most Python programs end in `.py`, however, it depends on the Integrated Development Environment (IDE) that you are using.

In this module, all the examples use the command-line mode to execute programs instantly and understand the process better. Once you complete a program, you can save it to be able to run it as a script later on.

2.2. The most used Python Integrated Development Environments (IDEs) per industry

In the previous section, we have mentioned Integrated Development Environments (IDEs), let us briefly explain what an IDE is. An IDE is essentially a software application that contains various facilities to computer programmers to develop their software such as a source code editor, build automation tools and a debugger. If it all seems to not make much sense right now, do not worry. Once we start practicing, you will have a better understanding of everything we have explained so far.

Depending on the industry, there are different Python IDEs that are preferred. Some of the factors that are taken into account when a company or organisation is choosing an IDE are: the intended use of the IDE such as Web Development, Data Science, Scripting or Quality Assurance; the Operating System that it uses (i.e., Linux/macOS, Windows, or mixed OS); whether the hardware is good or bad; and the level of the person that will be coding (e.g., beginner, intermediate or advanced).



Figure 7 - Most used IDEs per industry

(Adapted from <https://www.geeksforgeeks.org/top-10-python-ide-and-code-editors-in-2020/>)

2.3. Setting up a Python Integrated Development Environment (IDE)

In this module, you can use either Spyder (<https://www.anaconda.com/products/individual>) or Visual Studio Code (<https://code.visualstudio.com/>) as they are usually used for beginners to familiarise themselves with Python in a user-friendly environment.

However, if you are not ready to commit to an IDE, you also have the option of using online text editors such as the following:

- [Python.org](https://python.org)
- [Programiz](https://programiz.com)
- [Tutorialspoint](https://tutorialspoint.com)

We recommend using an IDE, because it is easier to keep track of your files and progress as you start coding.

2.3.1 Installing Visual Studio Code

1. Go to this website: <https://code.visualstudio.com/>

- Once the page loads, you will see the Download button. It automatically chooses your Operating System; however, you can click on the arrow next to it and change it according to your needs. Be sure to download the stable build!

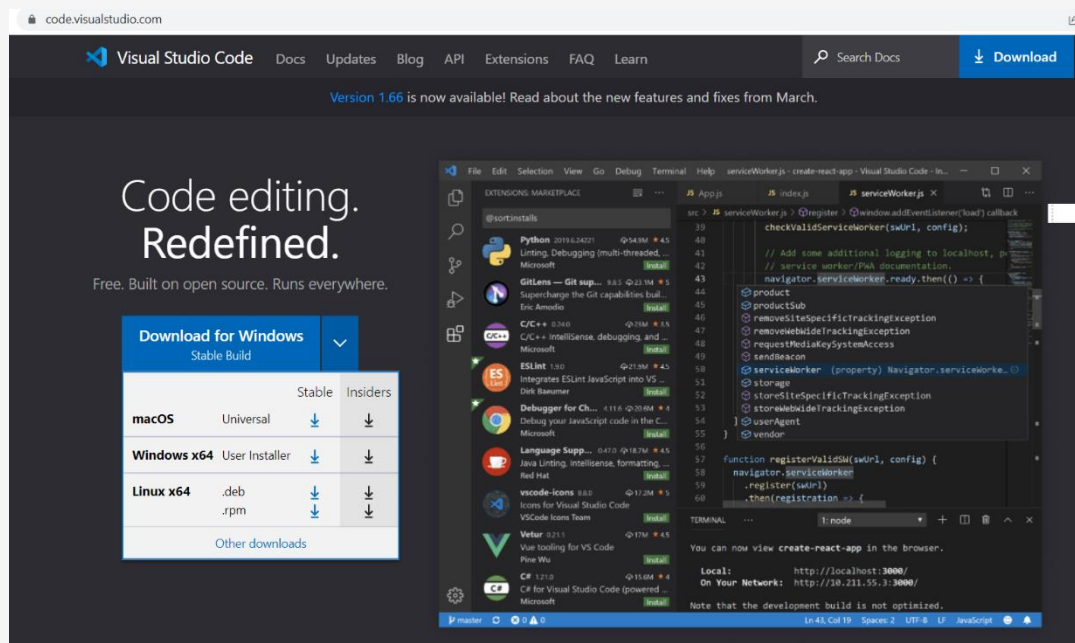


Figure 8 – Downloading Visual Studio Code

- When the download is completed, open the file and follow the setup instructions.

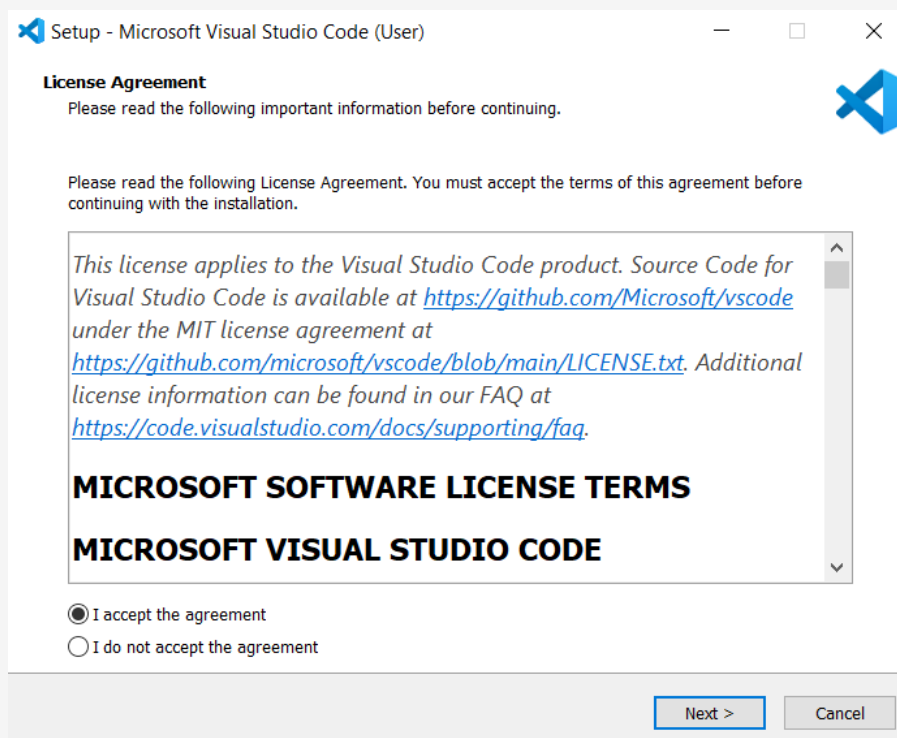


Figure 9 – Installing the IDE

4. Once the program has finished installing, you can open it or it will launch automatically.
5. Select File→New

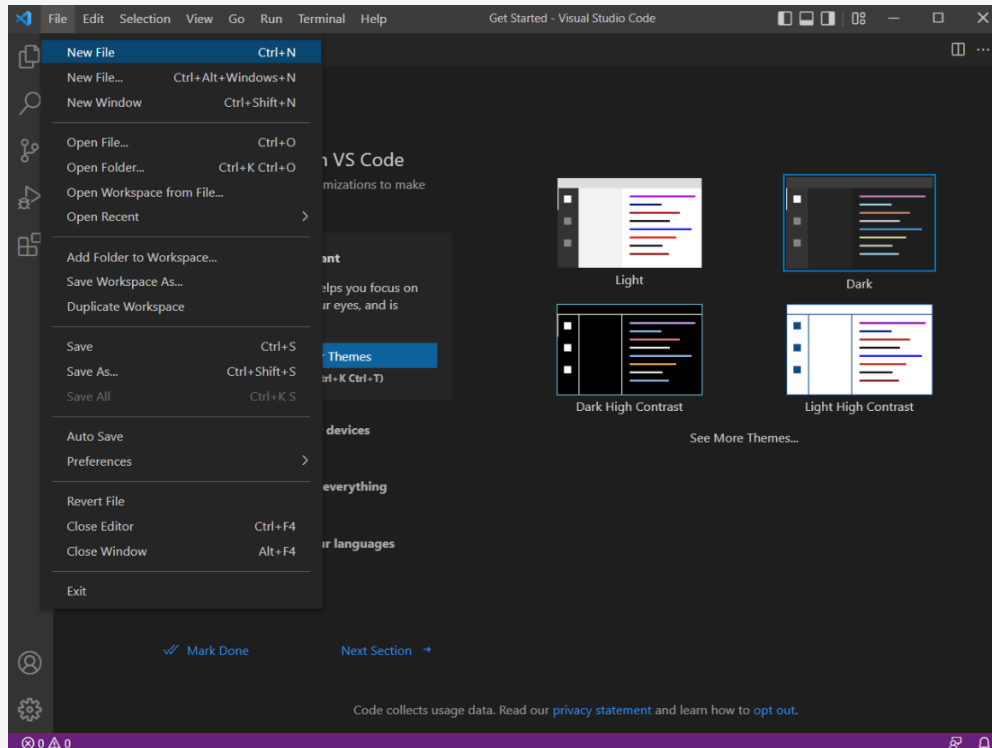


Figure 10 – Creating a new file in Visual Studio Co

6. Click on select a Language and pick Python

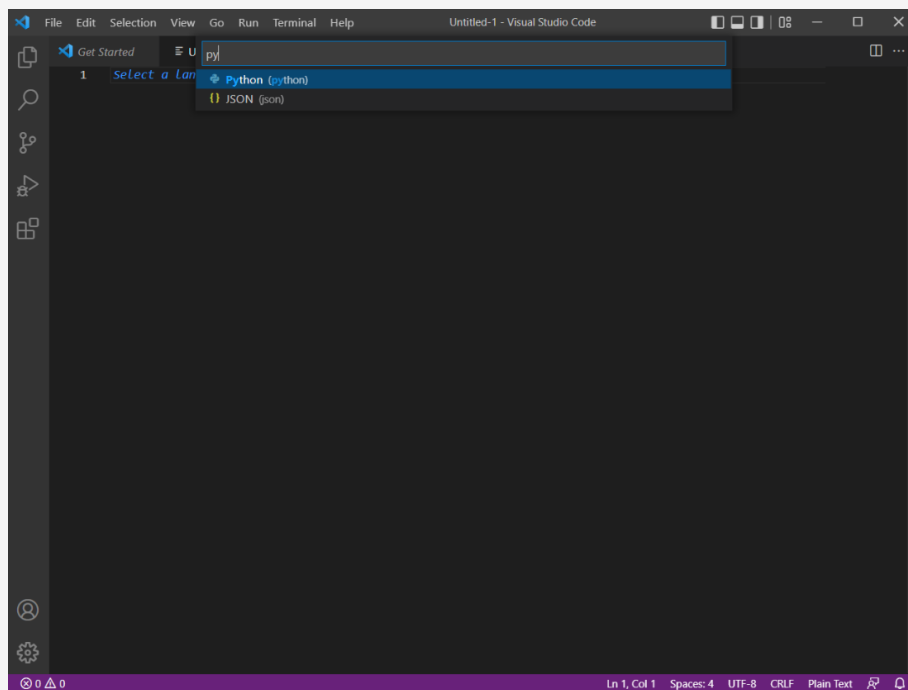


Figure 11 – Selecting the programming language

- Once you select Python, you will be redirected to install Python.



Figure 12 – Installing Python from Extensions

- Once you install Python, you are ready to start coding!

For more information, you can also consult their website on how to get started:

<https://code.visualstudio.com/docs/introvideos/basics>

2.3.2. Installing Spyder from Anaconda

- Go to this website: <https://www.anaconda.com/products/individual>
- Once the page loads, you will see the Download button. Depending on your operating system, choose the corresponding version. We recommend the 64-bit graphical installer.

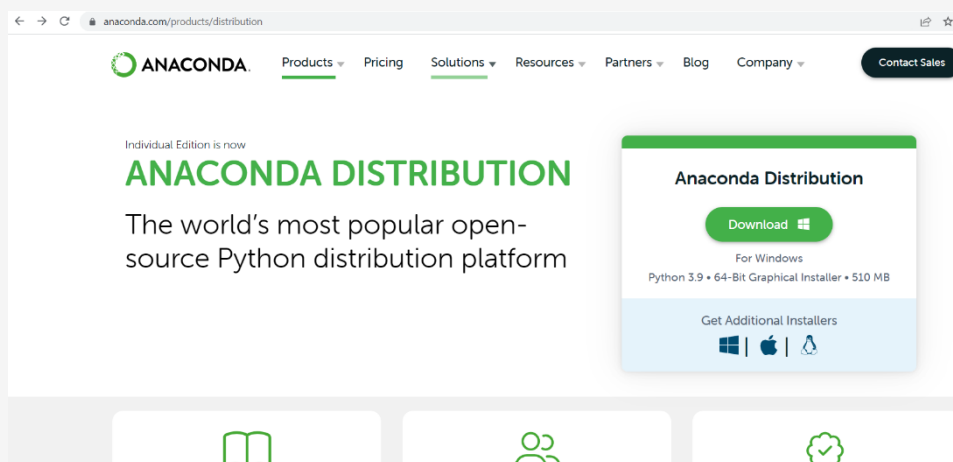


Figure 13 – Downloading Anaconda Distribution

- a. For more information on the steps required for Windows, follow this link:
<https://docs.anaconda.com/anaconda/install/windows/>
 - b. For macOS users, follow this link:
<https://docs.anaconda.com/anaconda/install/mac-os/>
 - c. For Linux users, follow this link:
<https://docs.anaconda.com/anaconda/install/linux/>
3. When the download is completed, open the file, and follow the setup instructions.

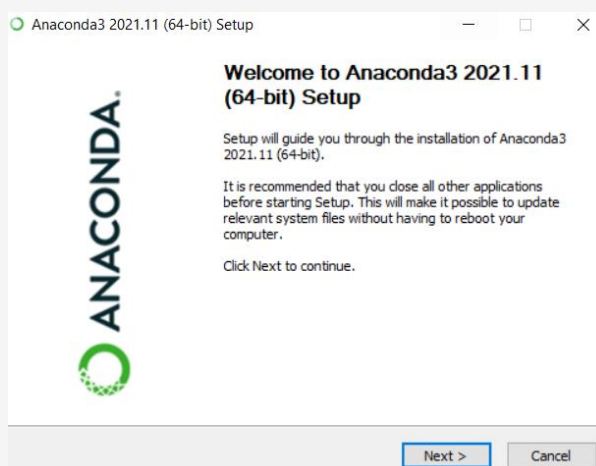


Figure 14 – Installing Anaconda

4. Once the program has finished installing, you can open the Anaconda Navigator.
5. Once the Navigator has loaded, select the Spyder application and click Launch. If it is not installed, click the Install button to install Spyder.

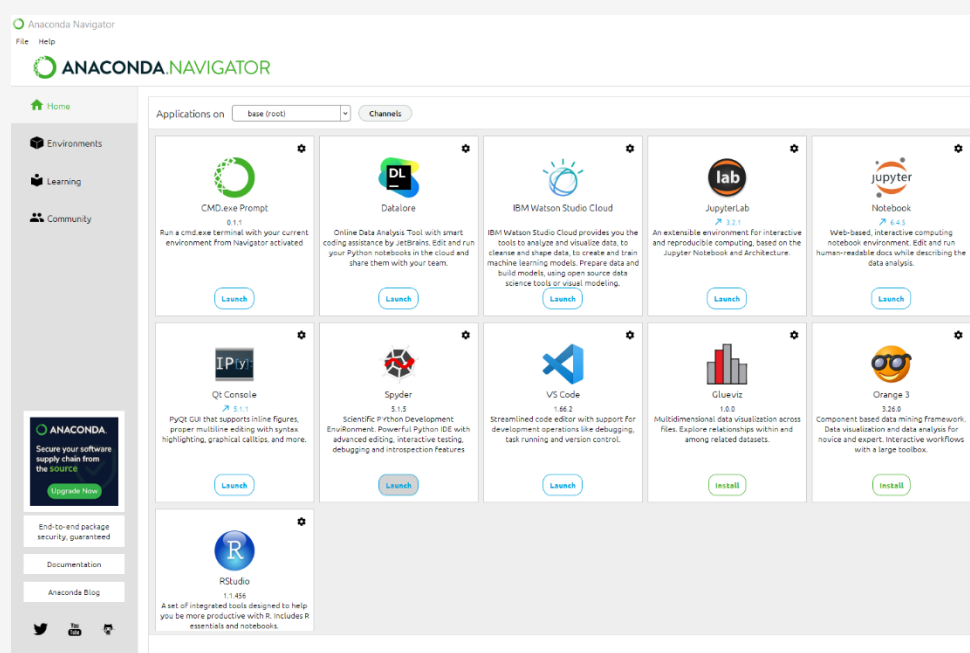


Figure 15 – Anaconda Navigator

6. When you open Spyder, you can select to take a tour or dismiss.

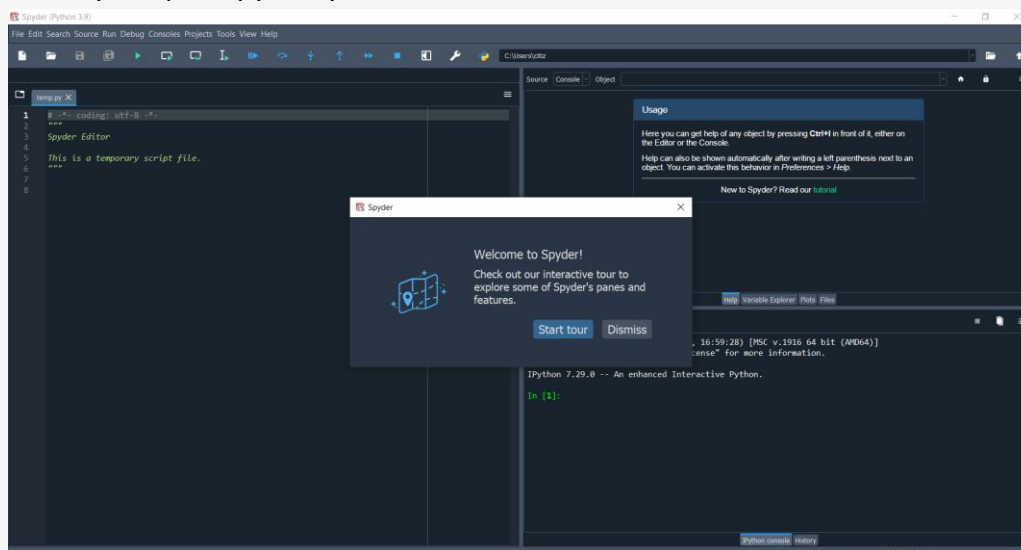


Figure 15 – Opening Spyder for the first time

7. Now you can start coding!

For more information, you can also consult the Anaconda starter:

<https://docs.anaconda.com/anaconda/user-guide/getting-started/>

We are sure you were able to complete this step easily!

2.4. Running Python in Command line

Now it is time to run our first program. You might be wondering if it is that simple. The answer is: Yes, it is!

Simply type the following:

```
print("Hello, World!")
```

Now, you can run it. It should print on the console: **Hello, World!**

3. Basics to programming in Python

- ⇒ **Number of participants:** 1-10 per facilitator
- ⇒ **Duration:** 8.00h
- ⇒ **Teaching methods:** Presentation, Guided instruction, Experiential learning
- ⇒ **Required materials:** Presentation, Computers (1 per participant) and stable internet connection

After you have successfully run your first program, we will continue to learn more about the basic functions of Python and the logic behind them.

3.1. The Basics: Data Types (Basic and Complex)

Data types are an important concept in the programming world since they determine how the data can be manipulated. The built-in data types in Python fall into the following categories:

General groups of data types	Specific data types used in Python	Examples
Text	str	"Hello, World!" (a series of strings, always use quotation marks)
Numeric	int, float, complex	3 (integer-int), 3.2 (float – includes decimals), 2j+3 (includes both characters and numbers)
Sequence	list, tuple, range	["apple", "cherry", 1, 1] (list; sequence of elements of any type), () (tuple; ordered collection), range(5) (iterates over sequence of number, has a starting and ending point (5))
Mapping	dict	{"brand": "Ford"} (dictionary; unordered collection of key-value pairs)
Set	set, frozenset	{"apple", "cherry", 1} (set; unordered collection of no duplicate items of any type separated by commas)
Boolean	bool	True or False
Binary	bytes, bytearray, memoryview	used to manipulate binary data

Table 2 – The different data types of Python with examples (adapted from https://www.w3schools.com/python/python_datatypes.asp)



These are all the built-in data types that exist in Python. We will go through all of these except the last group, binary, since the others are most useful for beginners to understand.

3.2. Variables, expressions, and statements

Variables

Variables essentially contain data that we assign to them in the form of names to make our code more readable and be able to utilise a particular variable more times than one.

We write the name that we want to assign to our variable, use the equal sign (=), and put the value that we want to store on the right side of the equal sign.

Example:

```
x = 5
print(x)
```

Here, variable x equals 5 and print(x) prints the value contained in x (i.e., 5).

***Keep in mind that Python is case-sensitive, therefore x and X will not be considered the same variable.**

Remember the data types that we just saw, you can write the following to see the data type of the variable x that we just created:

```
print(type(x))
```

The console will show: **int**

Suppose that you want to create 3 variables where one contains an integer, another a float and another a string. You would write the following:

```
age = 35
name = "John"
price = 12.99
```

Now if we try to run this, the console will not show anything. You might be thinking why that is, it is simply because we have not called the variable, but we only assigned a value to it. To see what our variable contains, we need to use **print (variable name)** to instruct Python to display it. Otherwise, it will not.

Example:

```
print(age)
print(name)
print(price)
```

Or you can print them all together like this:

```
print(age, name, price)
```



We can give any name to our variable, and we are not limited to only one word.

However, you should not use too long variable names because it is not convenient for you or the reader.

Try to keep the variables' names short and understandable. If you want your variable name to be two separate words, simply use underscore in between, i.e., `my_age`.

***Keep in mind that Python does not allow variable names to start with numbers or include special characters such as @, #, \$, etc.**

There are also some other keywords used in Python that cannot be used as variable names, since they are reserved for specific functions. These keywords are:

<code>and</code>	<code>def</code>	<code>exec</code>	<code>if</code>	<code>not</code>	<code>return</code>
<code>assert</code>	<code>del</code>	<code>finally</code>	<code>import</code>	<code>or</code>	<code>try</code>
<code>break</code>	<code>elif</code>	<code>for</code>	<code>in</code>	<code>pass</code>	<code>while</code>
<code>class</code>	<code>else</code>	<code>from</code>	<code>is</code>	<code>print</code>	<code>yield</code>
<code>continue</code>	<code>except</code>	<code>global</code>	<code>lambda</code>	<code>raise</code>	

Table 3 - Reserved Keywords in Python (Adapted from Downey et al., 2002, p. 15)

Statements

We have seen two statements used in Python so far: print and variable assignment. The commands given to the Python interpreter to be executed are called statements.

As we have already mentioned, statements, such as variable assignment, do not produce a result. However, the print statement produces a result, which is the value of the variable.

A sequence of statements is considered a script that either produces results or not, depending on the type of statements used. The results are displayed based on the order of the statements.

For example, if you type the following statements:

```
print(name)
height_cm = 1.75
print(age)
```

You would get as output:

John

35

The height variable is only saved as a variable but does not produce any result.

Expressions

There are different kinds of expressions used in Python. Expressions involve a combination of different variables, operators and values (operands) that produce another value as output.

As an example, let's try to add two numbers without assigning them variable names or use the print statement:

```
2+2
```

Output:

Here, we can see that Python will not produce an output. Therefore, it is necessary to have variables, operators and values to have an expression.

If you want to have a result from 2+2, then you need to either include the print statement or assign a variable name and print it.

Example:

```
print(2+2)
addition = 2+2
print(addition)
```

In the example of 2+2, we are asking Python to evaluate the expression. But if we simply wrote 2 and 2 in separate lines without the operator (+), no result would be produced. Even if the statement is considered legal, it does not produce an output.

Consider the following script:

```
56
4.5
"John"
print(3+2)
```

Here, the only output you would get is 5 from the last line of 3+2. What can we add to the script to show all the expressions on the console as output?

Hint: we have already used it multiple times by now

Operators

It is time to explain some of the operators used in Python, such as the addition sign (+) that we used earlier. Operators are the special symbols that represent mathematical computations, which are the following:

addition (3+20) subtraction (x-6) multiplication (3*5) division (10/5) exponentiation (4**3)



You can also use variable names that contain integers or floats, instead of plain integers to perform operations.

Example:

```
course_grade1 = 15
course_grade2 = 18
coursesum = course_grade1+course_grade2
print(coursesum)
```

Output: 33

To find the average grade based on the 2 course grades, we need to add them and then divide them by 2. How do you think this can be done? Take a minute to think about it.

Answer:

```
courseavg = (course_grade1+course_grade2)/2
print(courseavg)
```

Output: 16.5

This example also demonstrates that Python recognises the order of operations, which dictates that parenthesis is of the highest precedence instead of division that is located outside of the parenthesis.

In Python, the precedence is as follows: 1) parenthesis, 2) exponentiation, 3) multiplication/division and 4) addition/subtraction.

It should be noted that when the operators hold the same precedence, they are evaluated from left to right. For instance, consider the following operation: $5*30/60$. This equals to 2.5, which shows that the multiplication happens first (=150) and then the division that gives the end-result.

Practice: https://www.w3schools.com/python/exercise.asp?filename=exercise_variables1

Operators that work on strings

Even if a string is considered to be a number (i.e., '15'), you generally cannot perform the above-mentioned operations on strings.

However, the addition (+) and multiplication (*) operators have a different effect when applied on strings. If you use addition between two variables that contain strings, they will be concatenated. This means that they will be joined as one sequence of strings.

Example:

```
first_name = 'Emma'
last_name = 'Smith'
print(first_name + last_name)
```



Output: Emma Smith

*Keep in mind that if the quotation marks and the strings do have any space between them, you will end up “EmmaSmith” instead of “Emma Smith”. Therefore, spaces are also part of strings.

When used on strings, the multiplication (*) operator performs repetition.

Example:

```
print("First"*3)
```

Output: FirstFirstFirst

For this to work, you need to use an integer to specify how many times to repeat the string.

Comments

As programs grow, it becomes more difficult to track down and make sense of the algorithms used to produce an end-result. Thus, comments are useful to explain the rationale and process of the program in natural language. Think of comments like adding notes wherever needed to make your code more readable, you just have to add a hashtag (#) symbol for small comments.

Example:

```
#calculating the average grade of students based on the course grade
courseavg = (course_grade1+course_grade2)/2
```

You can also add a comment on the same line as the code:

```
print(first_name + last_name) #concatenating two strings
```

Whatever you write after hashtag (#) symbol is ignored and is not executed as code by the program.

Another option that can be used for larger parts of text is adding 3 quotation marks (""") before and 3 (""") after the text is finished.

Example:

```
"""
I can add as much text as I like to explain a function
"""
```

The use of comments is a useful practice for any professional that works with coding.

Practice:

https://www.w3schools.com/python/exercise.asp?filename=exercise_comments1



3.3. Lists, Dictionaries and Tuples

In this section, we will see how to use lists, dictionaries and tuples in Python. As we have seen earlier in the data types section, list and tuples fall within the sequence group and dictionaries within the mapping group. All these data types have different features and qualities that allow for data manipulation in different ways.

3.3.1. Lists

Lists are initiated with the use of square brackets [] and contain a sequence of ordered items in one variable.

Example:

```
adj = ["smart", "beautiful", "stunning"]
```

The items included in a list are ordered, mutable (i.e., changeable), and allow duplicate values.

Each item in a list is identified by an index. The index starts from [0] and goes up by one each time; thus, the first item on the list has an index [0] and the second one [1] and so on. It should be noted that their order cannot be changed. When new items are added, they will be put at the end of the list.

Also, lists are mutable, which means that items can be modified, removed or added after the list has been created. Since they allow duplicate values, you can find a value more than once in a list.

Example:

```
fruits = ["apple", "banana", "apple", "grapefruit"]
```

Lists can also contain a combination of integers, strings or boolean values:

```
employee1 = ["John Smith", 35, "male", 25000, True]
```

If you want to determine the number of items contained in a list, you can use the `len()` function:

```
print(len(employee1))
```

*** Keep in mind that the number of parentheses need to match from the beginning to end for the code to be executed.**

If you notice the code in this example, we have one beginning at the `len()` function and one within the list, so that is why you see two closing parenthesis at the end of the code.

You can also use the `list()` function to initiate a list:



```
new_list = list(("tomato", "cucumber", "peas", "peppers"))
print(new_list)
```

* Notice that here you need to add a double parenthesis when using the list() function to create a list.

Access items in lists

If you want to access an item in a list, you need to use their index number.

Let's use the employee1 list that we created earlier to get the age of the employee, which is the second item in the list, in the following example:

```
print(employee1[1])
```

* Do you remember why we are using 1 instead of 2 as the index number for the second item on the list? It is because the index starts from 0 in lists.

If you have a long list of items and you want to get the last item from the list, you can use negative indexing. This means that it will use -1 to access the last item found in the list, -2 to get the second last item, and so on.

Example:

```
print(employee1[-1])
```

You can also specify a range of indexes that indicates from which index number to start and to which index number to end the range.

Example:

```
print(employee1[2:4])
```

In this example, it will start from the third item on the list and end on the last one since we only have 5 items on the list. Always remember that the numbering starts from 0.

If you decide that you do not want to specify the beginning index number, the range will start from the first item found in the list:

```
print(employee1[:4])
```

You can also only specify the beginning index number without an ending index number:

```
print(employee1[1:])
```

Remember when we introduced negative indexing, you can also use that to access items on a list from the end of the list:

```
print(employee1[-4:-1])
```

This will get us the fourth item from the end until the last one found in the list.



You can also check if an item exists in the list. From the adjectives list that we created earlier, let's say that we want to check if beautiful is included in the list:

```
adj = ["smart", "beautiful", "stunning"]
if "beautiful" in adj:
    print("Yes, beautiful is included in the list")
```

Add items to lists

If you want to add new items to the end of the list, you can use the **append()** method. Let's say that we want to add a new adjective to the adjectives list, we will do the following:

```
adj.append("innovative")
print(adj)
```

You also have the option of adding new items by specifying the index number. You can do this by using the **insert()** method:

```
adj.insert(1, "innovative")
print(adj)
```

As you can see here, you first specify the index number and then the item that you want to add separated by a comma.

Another method that can be used when adding items from another list to the current list is **extend()**. Let's say that we have a general list of words and we want to add the adjectives into the general list of words that we have:

```
words = ["I", "am", "good", "you", "are", "nice"]
words.extend(adj)
print(words)
```

Modify lists

If you want to modify a specific item found in a list, you need to refer to its index number. As an example, we will use the employee1 list to modify the age of the employee:

```
employee1 = ["John Smith", 35, "male", 25000, True]
employee1[1] = 36
print(employee1)
```

You can also change multiple items in a list within a specified range. You can do this by defining the new items to be added and refer to the index numbers that you want to be modified.

For instance, we are going to modify the first three items of the employee1 list, because that employee is no longer working in that company:




```
employee1[0:2] = ["Ella Smith", 30, "female"]
print(employee1)
```

Remove items from lists

If you have decided to remove a specified item from a list, you can use the **remove()** method. Let's say that we want to remove the boolean value from the employee1 list:

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee1.remove(True)
print(employee1)
```

You can also remove an item by specifying its index number with the **pop()** method:

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee1.pop(4)
print(employee1)
```

If you do not specify an index number, then the last item found in the list will be removed:

```
employee1.pop()
print(employee1)
```

As an alternative, you can use the **del** keyword to remove an item with a specified index:

```
del employee1[1]
```

Also, you can use **del** to delete the entire list:

```
del employee1
```

However, you might want to empty the contents of the list. In this case, you can use the **clear()** method:

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee1.clear()
print(employee1)
```

Copy lists

There are two ways that you can copy a list with a new variable name. The first one is by using the **copy()** method:

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee2 = employee1.copy()
print(employee2)
```

The second way that this can work is by using the **list()** method:

```
employee2 = list(employee1)
print(employee2)
```



Sort lists

If you want to sort a list in alphanumerical order, either descending or ascending, you can use the `sort()` method.

***Note that the `sort()` method will list items in ascending order by default, if not specified otherwise.**

Let's use the words list to sort it in alphabetical order in this example:

```
adj = ["smart", "beautiful", "stunning"]
adj.sort()
print(adj)
```

Another example that includes a numerical list:

```
monthly_income= [5000, 1000, 2500, 1300, 1200, 1500, 2300]
monthly_income.sort()
print(monthly_income)
```

These two examples will list in ascending order, i.e., from smaller to bigger or a to z according to the data types.

If you want to sort a list in descending order, you need to specify it in the parenthesis by using the keyword `reverse = True`. Remember when we mentioned booleans as data types, in this case, it is used to enable the reverse option by stating that it is True since the default is False.

Example 1:

```
adj = ["smart", "beautiful", "stunning"]
adj.sort(reverse=True)
print(adj)
```

Example 2:

```
monthly_income= [5000, 1000, 2500, 1300, 1200, 1500, 2300]
monthly_income.sort()
print(monthly_income)
```

Sort is by default case sensitive, which means that all capital case letters are sorted before the lower-case letters.

In this example, you might get some surprising results because of this:

```
words = ["I", "am", "good", "you", "are", "nice"]
words.sort()
print(words)
```

To overcome this issue, you can use the case insensitive sort function, which is `key = str.lower`. Here, it is using the data type string to make it lower case.

Example:



```
words = ["I", "am", "good", "you", "are", "nice"]
words.sort(key=str.lower)
print(words)
```

In addition, you can use the **reverse()** method to reverse the order of a list regardless of its alphabetical order:

```
words = ["I", "am", "good", "you", "are", "nice"]
words.reverse()
print(words)
```

Join lists

Remember when we saw the arithmetic operators used and their effect on strings. One of the methods to join lists is by using the **addition sign (+)**:

```
adj = ["smart", "beautiful", "stunning"]
words = ["I", "am", "good", "you", "are", "nice"]
all_words = adj + words
print(all_words)
```

Here, we created a new list by combining the two already-existing lists of adjectives and words. Regardless of the data types found in either list, you can combine them with + operator.

If you want to add items from one list to another, simply use the **extend()** method:

```
words.extend(adj)
print(words)
```

In this example, the words list is joined with the adjectives list. Therefore, the words list will now contain both lists.

Practice: https://www.w3schools.com/python/exercise.asp?filename=exercise_lists1

3.3.2. Tuples

Tuples also contain a sequence of items in one variable like lists. Similar to lists, tuples are also ordered. However, one key difference between lists and tuples is that tuples are immutable (i.e., unchangeable). Tuples are contained in round brackets ().

Example:

```
coordinates = (38.09, -77.06)
```

In the real world, tuples are commonly used as a dictionary without keys to store data since they are faster to iterate through and their items cannot be changed. Tuples also allow duplicate values.

If you want to see how many items are in a tuple, you can use the **len()** function:



```
print(len(coordinates))
```

When creating a tuple that contains only one item, you need to add a comma after the item. Otherwise, Python will not recognise it as tuple. We will use the **type** function to determine the difference between using the comma.

Example:

```
name1 = ("John",)
print(type(name1))

name2 = ("John") #not a tuple
print(type(name2))
```

Similar to lists, tuples can contain items of any data type either in separate tuples or as a combination like the example below:

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1)
```

An alternative way to create a tuple is by using the **tuple()** constructor:

```
coordinates = tuple((38.09, -77.06))
print(coordinates)
```

*** Note that when using the tuple constructor, you need to add double round brackets.**

Access tuples

Items in tuples use indexing, which means that each item in the tuple corresponds to a number. The indexing is based on the order of items and it starts from 0 for the first item and goes up by 1 each time.

To access a specific item, you need to use square brackets and specify the index number:

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1)
```

***Remember that index number 1 in the tuple corresponds to the second item, i.e., 34.**

You can also use negative indexing to access the last item of a tuple [-1] or the second last item [-2] and so on.

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1[-1])
```

This will display the "male" item in the tuple.

If you want, you can specify a range of indexes that indicates from which index number to start and to which index number to end the range.



Example:

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1[1:3])
```

In this example, it will start from the second item on the list and end on the last one, since we only have 4 items on the list. Always remember that the numbering starts from 0.

If you decide that you do not want to specify the beginning index number, the range will start from the first item found in the tuple:

```
print(personal_info1[:3])
```

You can also only specify the beginning index number without an ending index number:

```
print(personal_info1[1:])
```

Remember when we introduced negative indexing, you can also use that to access items on a list from the end of the list:

```
print(personal_info1[-3:-1])
```

This will get us the fourth item from the end until the last one in the tuple.

You can also check if an item exists in a tuple, similar to what we saw in lists. Let's say that we want to check if beautiful is included in the adj tuple:

```
adj = ("smart", "beautiful", "stunning")
if "beautiful" in adj:
    print("Yes, beautiful is included in the adj tuple")
```

Add, modify, and remove items from tuples

Technically, you cannot add or remove items from tuples as they are immutable (i.e., unchangeable). However, there is a workaround. You just need to convert the tuple into a list, modify it and then convert it back into a tuple.

Let's see an example to understand this a bit better:

```
personal_info1 = ("John Kent", 34, True, "male")
#here we create a new variable that converts the tuple into a list
modify_pinfo1 = list(personal_info1)
modify_pinfo1[1] = 35 #modify/change the item that we want
personal_info1 = tuple(modify_pinfo1)
print(personal_info1)
```

If you want to add items into a tuple, there are two ways that you can use. The first one follows the same logic as the example we saw to modify an item found in a tuple.

Example:



```
personal_info1 = ("John Kent", 34, True, "male")
modify_pinfo1 = list(personal_info1) #convert tuple into list in a new variable
modify_pinfo1.append("1 Charming Street") #adding new item
personal_info1 = tuple(modify_pinfo1) #convert it back
```

The second way to add a new item is by adding a tuple to another tuple. You can create a new tuple that contains the item that you want to add to the existing tuple and simply add it:

```
personal_info1 = ("John Kent", 34, True, "male")
modify_pinfo1 = ("1 Charming Street",) #creating new tuple, do not forget the comma
personal_info1 += modify_pinfo1 #adding new tuple into the existing one
print(personal_info1)
```

Here, we will see how we can remove items from tuples. The same logic applies as in the previous examples of add or modify items in tuples.

Example:

```
personal_info1 = ("John Kent", 34, True, "male")
modify_pinfo1 = list(personal_info1) #convert tuple into list in a new variable
modify_pinfo1.remove("male") #delete item male
personal_info1 = tuple(modify_pinfo1) #convert back
print(personal_info1)
```

In the following example, we will see how you can delete the tuple by using the del keyword:

```
personal_info1 = ("John Kent", 34, True, "male")
del personal_info1 #it is now deleted
print(personal_info1) # this will raise an error since we have deleted it
```

When creating a tuple, we are assigning values or items to be contained in it. This process is also called “packing” a tuple:

```
personal_info1 = ("John Kent", 34, True, "male")
```

In Python, you also have the opportunity to extract the values into variables, which is called unpacking:

```
personal_info1 = ("John Kent", 34, True, "male")
# assigned separate variable name to each tuple item
(name, age, employed, sex) = personal_info1
# printing each new item variable from tuple
print(name)
print(age)
print(employed)
print(sex)
```

*** Note that the variable names must match the items contained in the tuple, otherwise you can use an asterisk (*) to collect the rest as a list.**

Let's say that you want to extract only two variables from the tuple `personal_info1`, you can use the asterisk:

```
personal_info1 = ("John Kent", 34, True, "male")
(name, *demographics) = personal_info1
# all items after the first will be contained in the demographics variable
print(name)
print(demographics)
```

If you add the asterisk to another variable name instead of the last, then values will be assigned to the variable with the asterisk until the values left match the variables left:

```
personal_info1 = ("John Kent", 34, True, "male")
(name, *age_job, sex) = personal_info1
print(name)
print(age_job)
print(sex)
```

Join tuples

If you want to join two tuples, you can use the **addition operator (+)**:

```
personal_info1 = ("John Kent", 34, True, "male")
personal_info2 = ("Kylie Smith", 40, True, "female")
total_pinfo = personal_info1 + personal_info2
print(total_pinfo)
```

You also have the option of multiplying the content of a tuple for a number of times by using the **multiplication operator (*)**:

```
personal_info1 = ("John Kent", 34, True, "male")
general = personal_info1 * 2
print(general)
```

Practice: https://www.w3schools.com/python/exercise.asp?filename=exercise_tuples1

3.3.3. Dictionaries

Dictionaries fall within the mapping group of data types since they can store data in key:value pairs. They contain a sequence of pair elements which are ordered, mutable (i.e., changeable) and do not allow duplicate values. Dictionaries are marked by curly brackets {}.

*** Note that dictionaries in Python 3.6 and earlier versions were unordered, but from Python 3.7. they became ordered.**

An example of a dictionary can be for translating English into another language. Let's say we want to translate from English to Spanish:



```
eng2esp = {
    "hello": "hola",
    "good morning": "buenas dias",
    "good afternoon": "buenas tardes",
    "how are you": "como estas"
}
print(eng2esp)
```

Another option is to create an empty dictionary and add elements one by one:

```
eng2esp = {}
eng2esp["hello"] = "hola"
eng2esp["good morning"] = "buenas dias" # and so on
```

As we have mentioned, dictionaries do not allow duplicate values for their keys, i.e., “hello”, in the eng2esp dictionary that we created above.

If you want to know how many items are in a dictionary, you can use the **len()** function as we saw for lists and tuples:

```
print(len(eng2esp))
```

Once again, similar to tuples and lists, dictionaries can contain any type of data:

```
sofas_inv = { "sofas": 5,
              "colours": [ "red", "blue", "grey" ],
              "year": [2016, 2022, 2018],
              "sofabed": False}
```

As you can see here, we have int, strings, bool and list data types.

To determine the data type of the sofas variable, use **type()**:

```
print(type(sofas_inv))
```

Access items in dictionaries

If you want to look up what “hello” is in Spanish, then you can use the following:

```
eng2esp = {
    "hello": "hola",
    "good morning": "buenas dias",
    "good afternoon": "buenas tardes",
    "how are you": "como estas"
}
print(eng2esp["hello"])
```

This will print out: “hola”

When looking up items in a dictionary, use their key name within square brackets:

```
# here we created a variable to store the value "hola"
esp_hello = eng2esp["hello"]
```

Here “hello” is the key to the value “hola”.

An alternative method is to use the **get()** method to have the equivalent result:

```
esp_hello = eng2esp.get("hello")
```

If you want to see a list of all the keys that a dictionary contains, you can use the **keys()** method:

```
all_keys = eng2esp.keys()
print(all_keys)
```

Any changes made on the dictionary `eng2esp` will be reflected on the `all_keys` list that we created:

```
eng2esp["bye"] = "adios"
print(all_keys) #after the addition
```

Also, you have the option of getting all the values contained in a dictionary with the **values()** method:

```
all_values = eng2esp.values()
print(all_values)
```

The same logic applies for the values in terms of changes. If we add a new value, it will be added to the list of values that we created.

Another option that you have is getting all key:value pairs by using the **items()** method:

```
all_pairs = eng2esp.items()
print(all_pairs)
```

Again, any changes made on the dictionary either in its keys or values will be reflected in the `all_pairs` list that we created.

If you are unsure whether a particular key exists in a dictionary, you can check:

```
if "hello" in eng2esp:
    print( "Yes, 'hello' is one of the keys of the eng2esp dictionary")
```

Add, modify, or remove items from dictionaries

To add items in a dictionary, there are two ways that you can use. The first one uses a new index key and its corresponding value to be added:

```
sofas_inv = { "sofas": 5,
              "colours": [ "red", "blue", "grey"],
              "year": [2016, 2022, 2018],
              "sofabed": False}
sofas_inv["material"] = "leather"
print(sofas_inv)
```

The second way is to use the **update()** method, where you need to specify the dictionary, or the iterable pair of key:value:

```
sofas_inv.update({"material" : "leather"})
print(sofas_inv)
```

***Note that the curly brackets after the parenthesis are needed to indicate the key:value pairs of the dictionary.**

If you want to modify items in a dictionary, the same options as described above are available. The first option is to refer to the key name to change a specific value. For example, if a sofa was sold, you can change the quantity of sofas in store:

```
sofas_inv = { "sofas": 5,
              "colours": [ "red", "blue", "grey"],
              "year": [2016, 2022, 2018],
              "sofabled": False}
sofas_inv["sofas"] = 4
```

The second option refers to the **update()** method:

```
sofas_inv.update({"sofas": 4})
```

There are 3 different ways to remove items from dictionaries. The first one uses the **pop()** method:

```
sofas_inv = { "sofas": 5,
              "colours": [ "red", "blue", "grey"],
              "year": [2016, 2022, 2018],
              "sofabled": False}
sofas_inv.pop("sofabled")
print(sofas_inv)
```

The second way is to use the **del** keyword:

```
del sofas_inv["sofabled"]
print(sofas_inv)
```

The third way available removes the last item added in Python 3.7. However, in earlier versions, a random item will be removed instead.

Example:

```
sofas_inv.popitem()
print(sofas_inv)
```

If you want to completely delete a dictionary, you can use the **del** keyword:

```
del sofas_inv
print(sofas_inv) #Python will raise error since the dictionary no longer exists
```

Another option that you have is to simply empty the dictionary by using the **clear()** method:



```
sofas_inv.clear()
print(sofas_inv) # you will have an empty dictionary
```

Copy dictionaries

Remember when we talked about copying lists and we said that you cannot copy one item to another by using the equal sign (=) because the changes of one list will be done on the other one. Similar to lists, dictionaries should not be copied that way. There are two ways to make a copy of a dictionary.

The first one is to use the **copy()** method:

```
new_inv = sofas_inv.copy()
print(new_inv)
```

The second way is to use the **dict()** function:

```
new_inv = dict(sofas_inv)
```

3.4. Conditionals and Loops

In this section, we will explain the logic and usage behind conditional statements and loops, which are considered essential and useful knowledge in general-programming languages.

- ⇒ Conditional statements use the statements **if**, **elif**, **else**.
- ⇒ **For** and **while** are used for loops.

Before we explain conditionals and loops in detail, we will learn some useful operators that are commonly used in these statements: boolean expressions (==) and logical operators (and, or, not).

3.4.1. Boolean expressions

A boolean expression is used to determine if an expression or statement is either true or false. There are two ways to write boolean expressions, the first one uses the == operator to compare two values and return a boolean value:

```
print(8 == 8)
```

Output:

True

```
print(9 == 8)
```

Output:

False

In both statements, the == operator was evaluating if the two integers were the same (i.e., had the same value). As you can see, the first statement yielded True and the second statement yielded False.



The == is one of the many comparison operators used in Python. The rest of the comparison operators are shown below:

x > y	x is greater than y
x < y	x is less than y
x >= y	x is greater than or equal to y
x <= y	x is less than or equal to y
x != y	x is not equal to y

You have probably come across these operators before, however, some of these symbols have different functions in Python.

For example, if you want to see if a value is the same as another, you would not use a single equal sign (=) because that would assign the value to the variable that is on the left-hand side.

To compare values in boolean expressions, you need to use **double equal sign (==)**.

Also, if you want to check whether a value is **greater or equal (>=)** to another value, you need to put the **greater (>)** or **less (<)** than sign first and then the **equal sign (=)**.

These are important distinctions in Python that you should remember to avoid errors or unwanted surprises.

Let's see some examples of these comparison operators with the keyword **if**:

```
x = 55
y = 500
if y > x:
    print("y is greater than x")
```

In this example, we are comparing the variables x and y to see if y is greater x. Since we already know that y is greater than x, we choose to print that as output.

*** The indentation that occurs in the second line (i.e., the whitespace found in the beginning of the line) is used to tell Python that this statement is part of that particular block of code. Indentation is an important part of the Python syntax to avoid raising errors.**

Another keyword used in conditional expressions is **elif**, which is used as an alternative if the first condition is not true. Let's see an example of comparison operators with the keyword **elif**:



```
x = 55
y = 55
if y < x:
    print("y is greater than x")
elif x == y:
    print("x is equal to y")
```

In this example, we have the first condition that checks if y is greater than x. Since it is not, the program moves to the second condition with **elif (a combination of else and if)** that checks if x is equal to y, which is true and prints the corresponding statement.

The next keyword used in conditionals is **else**, which is the last option available if the preceding conditions are not true.

Let's see an example of **else** to understand how it works:

```
x = 500
y = 55
if y > x:
    print("y is greater than x")
elif x == y:
    print("x is equal to y")
else:
    print("x is greater than y")

if x > y: print("x is greater than y")
```

Here, you can see that the **if** condition is not true, then it moves on to the second condition that is also not true and ends up with the final available option in **else**, which is true and prints the corresponding statement.

Also, if you only have a short statement to execute, you can write it in only one line:

```
if x > y: print("x is greater than y")
```

You can also do the same thing for a combination of **if** and **else** statements:

```
print("x is greater than why") if x > y else print("y is greater than x")
```

Another option is to include multiple **else** statements on one line:

```
print("X") if x > y else print("=") if x == y else print("Y")
```

3.4.2. Logical operators

Python includes 3 logical operators, as we have mentioned in the beginning of this section, **and**, **or**, and **not**. These operators have analogous uses in natural languages that combine one or multiple conditional statements.



Let's see how **and** can be used with conditional statements:

```
x = 500
y = 55
z = 800
if x > y and z > x:
    print("Both conditions are true")
```

Let's see an example of the **or** operator that combines conditional statements:

```
x = 500
y = 55
z = 800
if x > y or z > y:
    print("One of the two conditions is true")
```

Our next example is focused on the **not** operator:

```
x = 500
y = 55
if not y == x:
    print("x and y are not equal")
```

3.4.3. Nested if statements

The statements that contain an **if** statement inside **another if** statement are referred to as **nested if statements**. Let's check out the following example to understand this:

```
y = 50
if y > 20:
    print("y is above 20, ")
    if y > 30:
        print("and above 30 ")
    else:
        print("but not above 30")
```

Similar to functions, **if** statements should not be empty. However, if for a particular reason you have an **if** statement without any content, use the **pass** statement to avoid getting errors from Python:

```
x = 500
y = 55
if x > y:
    pass
```

Practice: https://www.w3schools.com/python/exercise.asp?filename=exercise_ifelse1

3.4.4. Loops

Python has two main commands to create loops: **while** and **for**.

The **while** loop is used when you want to keep executing as long as the condition is true. As an example, let's create a **while** loop that prints *i* as long as it is less than 10:



```
i = 1
while i < 10:
    print(i)
    i += 1
```

In this example, we are saying to Python as long as *i* is less than 10, continue to print *i* and increment by 1 each time the loop repeats.

***Keep in mind that if you do not increment *i*, the while loop will keep running without stopping and you will end up in an infinite loop.**

You can also use the **break** statement if we want to stop the loop even if the condition is true:

```
i = 1
while i < 10:
    print(i)
    if i == 5:
        break
    i += 1
```

Another available statement in loops is **continue** that will stop the current iteration and move to the next one:

```
i = 1
while i < 10:
    i += 1
    if i == 5:
        continue
    print(i)
```

The **else** statement that we saw in the conditional statements can also be used in loops:

```
i = 1
while i < 10:
    print(i)
    i += 1
else:
    print("i is no longer less than 10")
```

Practice:

https://www.w3schools.com/python/exercise.asp?filename=exercise_while_loops1

Loops that use the **for** keyword iterate over a sequence of items, which can be contained in lists, tuples, dictionaries, sets, or strings.

Let's see an example of this:

```
nouns = ["table", "book", "chair", "house"]
for noun in nouns: # for each noun in the nouns list
    print(noun)
```



In this kind of loop, you do not need to specify the index variable beforehand. An index variable is, in this case, the singular version of nouns as the value to be fetched (i.e., each item in the nouns list).

The **for** loop can also iterate over a string as sequences of characters:

```
for x in "book": # for each character in the word book
    print(x)
```

The **break** statement that we saw earlier can also be combined with a for loop:

```
nouns = ["table", "book", "chair", "house"]
for noun in nouns: # for each noun in the nouns list
    print(noun)
    if noun == "chair":
        break
```

In this example, when the loop encounters the noun "chair", it will stop iterating before going through all items in the list.

If you had put the **break** statement prior to the print part, what do you think would happen?

Try it for yourself:

```
for noun in nouns: # for each noun in the nouns list
    if noun == "chair":
        break
    print(noun)
```

Another statement that we saw earlier is the **continue** statement, which interrupts the current iteration to move on to the next one:

```
for noun in nouns: # for each noun in the nouns list
    if noun == "chair":
        continue
    print(noun)
```

The range function is quite useful when you want to specify the number of times that you want the iteration to occur. The range function's default starting point is 0, which increases by 1 at each iteration and ends at a specified number:

```
for x in range(10):
    print(x)
```

*** Keep in mind that because the range starts from 0, it will stop at the number 9. If we actually want it to go up to 10, we would use range(11).**

You have the option of specifying the starting and ending range:

```
for x in range(2, 10):
    print(x)
```

Here, the range starts from 2 and ends at 9, since 10 is not included in the range.



Another option that you have is to adjust by how much the number increases at each iteration:

```
for x in range(2, 40, 2):
    print(x)
```

In this example, we specified the range to start from 2 until 40 and increase by 2 each time. Therefore, the number will stop at 38, since it cannot increase by 2 from 38 onwards.

In the **for** loop, you can also use the *else* keyword to specify what happens when the loop ends:

```
for x in range(10):
    print(x)
else:
    print("Done!")
```

Let's try to use **if** and *else* in a **for** loop:

```
for x in range(10):
    if x == 8: break
    print(x)
else:
    print("Done!")
```

Do you think the **else** statement will be executed? Why or why not? Try it to find out!

Remember when we talked about **nested if** statements, you can also have nested loops:

```
words = ["I", "am", "you", "are", "working"]
nouns = ["book", "house", "person", "love"]
for word in words:           # outer loop
    for noun in nouns:       # inner loop
        print(word, noun)
```

As you can see, the inner loop will be executed one time during each iteration of the outer loop.

Similar to functions and **if** statements, **for** statements should not be empty. However, if for a particular reason you have a **for** loop without any content, use the **pass** statement to avoid getting errors from Python:

```
for word in words:
    pass
```

Practice: https://www.w3schools.com/python/exercise.asp?filename=exercise_for_loops1

3.5. Understanding the flow of execution through functions

So far, we have seen a number of built-in functions used in Python such as `print()`, `type()`, `dict()`, `len()`. Each function has one specific purpose and needs us to specify which variable we want it to execute.



Example:

```
x = 5
print(x)
```

In the print function, we need to specify in the parenthesis the variable `x`, which is the one that we want to print. Whatever variable or variables are contained in the parenthesis are called **arguments or parameters**.

These two terms are usually used interchangeably, but **parameters** are the variables listed inside the parenthesis of a function, e.g., `print(x)`, whereas **arguments** are considered to be the values that are sent to the function when we call it.

Even though Python's built-in functions are quite useful, they cannot always solve all the problems that we want them to solve. Therefore, we have the option to create new functions to solve specific problems, which is considered one of the most beneficial aspects of a general-purpose language.

A function contains a sequence of statements that produce a desired operation. The syntax used for a function is the following:

```
>> def name(arguments):
    statements
```

Def means definition, it is essentially defining the name of the function along with its argument(s) to produce a desired result when called. The built-in functions have already been defined and thus, we cannot see the statements that they contain, only the produced result or output.

When creating your own function, you are free to use any names that you want except the Python keywords that we mentioned in earlier sections (Section 3.2.). The arguments used in the parenthesis of the function provide the information required for the function to work.

Let's see an example to understand the process followed by a function:

```
def my_function():
    print("Hello, World!")
```

Here we have created a function without any required arguments that will print the text specified in the second line of the code.

Remember when we talked about indentation in the previous section? The same logic applies here where indentation indicates that the code to be executed is **local** to the function.



Therefore, writing the statement `print("Hello, World!")` outside the function as we did in previous examples would be considered a **global** statement. However, now that is written within a function, it is considered local.

To call the function, we simply use its name followed by the parenthesis:

```
my_function()
```

The output of this function will be:

Hello, World!

Let's see another example with a specified argument inside the parenthesis of the function:

```
def my_function(country):
    print("I am from " + country)
```

When the function is called the country name is passed within the function to be printed as we instructed. The argument `country` can be specified when we call the function:

```
my_function("France")
my_function("Greece")
my_function("Germany")
```

Output:

I am from France

I am from Greece

I am from Germany

You can also specify the data type of the argument using the following syntax:

argument:datatype

Example:

```
def my_function(country:str):
    print("I am from " + country)
```

When calling the function, Python is going to expect the country to be in the form of a string. If it is not, it will raise an error.

```
my_function("France")
my_function(France) #will raise error
```

In addition, you can have more than 1 argument. Let's say that we wanted to specify the city and country:

```
def my_function(city, country):
    print("I am from " + city + ", " + country)
```

Notice that we included quotation marks with a comma to separate the city and country from each other in order to not have the two words printed without a space. Since we have



specified 2 arguments, we need to call exactly the number of arguments specified in the parenthesis, not more or less, for the function to work. Otherwise, it will raise an error.

```
my_function("Paris", "France")
```

Output:

I am from Paris, France

```
my_function("Paris") # this will raise an error in Python
```

Another option that you have if you do not know the number of arguments to be included in a function, simply add an asterisk (*) before the argument name in the function definition.

```
def my_function(*countries):
    for country in countries:
        print("I have visited ", country)
```

In this way, the function will receive a tuple of arguments and the items can be accessed accordingly.

*** These types of unknown number of arguments are called arbitrary arguments and are often referred to as *args in Python documentation.**

Here we specified 3 countries to be printed in separate sentences and in one sentence:

```
my_function("Ireland", "France", "Belgium")
my_function("Ireland, France, Belgium")
```

Output:

I have visited Ireland

I have visited France

I have visited Belgium

I have visited Ireland, France, Belgium

Another option available to you is to send arguments as pairs of key = value. In this way, the order does not matter.

Example:

```
def my_function(first, last):
    print("My name is " + first + last )
my_function(first = "Jane ", last = "Kent")
```

One more option available is the use of **two asterisks (**)** before the argument name in the function definition if you do not know how many keyword arguments will need to be

passed. In this way, the function will receive the arguments in dictionary form and will access them accordingly:

```
def total_words(**words):
    print(words, type(words))

total_words(paper = 6, I = 10, nice = 9)
```

As you can see, you can add new arguments when you call the function. Keep in mind that these types of arguments are called **arbitrary keyword arguments** and are often referred to as ****kwargs** in Python documentation.

Furthermore, one more option that you have is to set a default parameter value. This means that if we call the function without an argument, it will use the default value that we set:

```
def my_function(country= "United Kingdom"):
    print("I am from" + country)

my_function("France")
my_function("Malta")
my_function()
```

An argument can allow any data type as an argument (e.g., string, list, dictionary, etc.).

Functions can also return values by using the **return** statement:

```
def calc(x):
    return 3 * x

print(calc(5))
print(calc(8))
print(calc(9))
```

Since we have not specified the print function in our created function, we need to use it when we call the function in order to get the produced result.

Generally, functions should not be empty. However, if for a particular reason you have a function definition without any content, use the pass statement to avoid getting errors from Python:

```
def my_function():
    pass
```

Practice: https://www.w3schools.com/python/exercise.asp?filename=exercise_functions1

3.6. Putting the pieces together - How to build a program

So far, we have learnt a lot of different statements, keywords, methods, and functions used in Python. Now you might be wondering, how can we actually build a simple program? We will go through the process of building a small program step by step, so you can understand how to use what we have learnt.



We want to build a program that:

1. reads a text file,
2. creates a dictionary that contains all the words of the text file and their frequency,
3. allows users to write a word and display its frequency,
4. provide an informative message if a word does not exist.

This might seem overwhelming at first, but do not worry! We will go through it step by step.

Here, we will learn how to create recursive functions, which means that the functions will call each other to execute our program.

First, we need to call a module, which is called **string**. The built-in string module contains several functions that allow you to manipulate strings in Python, which we will use to remove all sets of punctuation later on.

There are many modules available in Python for a variety of purposes that you can call by simply using the keyword import:

```
import string
```

The first function will read the text file and create a dictionary:

```
def process_text(filename):
    dictionary = dict()
    fin = open(filename, 'r')
    for line in fin:
        process_line(line, dictionary)
    return dictionary
```

We always try to give a function name that we can understand, and the filename argument will be specified when we call it. The next line creates an empty dictionary. The **local fin variable** calls the **open** function to go open the file and read it. The **for** loop in the next line calls the next function that will process each line, create our dictionary, and return it.

It might sound a bit technical, but it is pretty simple. We essentially created a function that will create an empty dictionary, read the filename, and process each line of the file to create a dictionary.

Now, why do we want to process each line?

Because Python will differentiate between capital and lower-case words, as well as punctuation points next to words, even if the word is the same such as “Love” and “love” or “love,” and “love”.



Since we want to count the frequency of each word in the file, we need to make sure all words are lowercase and all punctuation points (i.e., commas, full stops, exclamation marks, etc.) are removed for the program to count correctly.

Let's write a function that processes each line of our file:

```
def process_line(line,dictionary):
    line = line.replace('-', '')

    for word in line.split():
        word = word.strip(string.punctuation + string.whitespace)
        word = word.lower()
        dictionary[word] = dictionary.get(word,0) +1
```

Again, here we are using a **for** loop to iterate through each line of the text.

- ⇒ First, we replace dashes with whitespace because they cannot be removed with the **string.punctuation** function.
- ⇒ Once we replace the dashes, we start by splitting the line into separate words and take each word to remove punctuations, whitespace and then make it lower-case.
- ⇒ After all the processing the word is added to the dictionary and **if the word exists you add 1, if not 0.**

So far, we have created two functions that read, edit the text and create a dictionary of the words' frequencies.

At this point you assign to the variable dictionary the function that processes the text, where you specify the text file's name (i.e., 'the_veldt.txt'). All the functions are connected based on the variable dictionary that will contain all the words found in a given text with their frequencies.

```
dictionary = process_text('the_veldt.txt')
```

Now, we are done with 2 out of the 4 actions that we want the program to do, we only have 2 more left.

Now, we want users to be able to write a word and if the word exists in the text to display its frequency, otherwise to inform the user that the word does not exist.

Let's write a function that will compare our user's input with the dictionary that we created:

```
def findwords(dictionary):
    for key,value in dictionary.items():
        if key == find_word:
            return(value)
    return("This word was not found in the text, please look for another word ")
```

This function is used to access the key and value of each dictionary item through a **for** loop. If the key (word) is the same as the word written by the user, then it returns the value (i.e.,

the frequency of a given word). Otherwise, it returns a statement that the word was not found and prompts the user to look for another.

```
while True:
    find_word = input("Please enter word to find its frequency, or type 'q' to quit: ").lower()
    if find_word != 'q':
        print(findwords(dictionary))
        continue
    else:
        break
```

Now for the last part of the code, we want the program to repeat until the user specifies to quit. For this part, we will use a **while** loop that is not in a function.

We simply use the **while True** loop to keep the program going. Be careful when using a **while True** loop because you might end in an infinite loop. **While True** signifies as long as this is true, continue to execute the program.

We assign a variable for the user input in order to be able to compare it with the keys of the dictionary (words) and whatever the user inputs will be converted into lower case when the program reads it. The reason for that is that a user might write a word in all capital letters and the program will not produce a result, because Python does not recognise that they are the same word, as we have mentioned Python is case-sensitive.

The next line says that if the input is not equal to 'q', then print the results of the function that retrieves the frequency and move on to the next iteration. Otherwise, if the user's input is **q**, it stops the program (i.e., break the loop).

Depending on who is writing the code, this small program could have been written in different ways. We have tried to incorporate as many of the things that we have learnt here, as well as introduce recursive functions and how to break your program into small and manageable pieces of code.

*** Keep in mind that when writing code, you should try to run each small part of your code to catch potential errors before you continue to the next part.**

Once you complete your program, you might need to go back and edit to remove some experimental/beginning parts of the code or consolidate multiple statements to make the program more compact and easier to read.

3.7. How to adjust a program to fit your needs

We have seen how to build a small program in Python, however, sometimes you might encounter programs that are already written, and you might need to make adjustments to the code to improve or change the end-result.

Depending on who wrote the code and how long it is, this can be a difficult and overwhelming procedure.



You can follow the same logic as when building a program, where you take small parts of the code to realise what each function does, if it is not clear.

Also, it is important to write comments when working with code as comments can be extremely helpful to other people reading your code and to you as well.

So, what do you do when there are no comments on a program?

First of all, run the program to see what its output is and notice the libraries or modules that are imported. Then, you should look for the starting point of the code and try to understand the flow of execution. Another helpful tip would be to look at the variables that the program holds and their usage throughout the program. After you have a good understanding of how the program works, you can start editing.

*** Be careful when you try to edit someone else's function because you might end up breaking it**

A good practice when adjusting a program is not to use a specific number of variables, but rather use ***args** and ****kwargs** to make your code more flexible and adaptable when another user might need to adjust it. We have seen the use of these two variables earlier in the module.

As a reminder, ***args** allows you to pass a varying number of positional arguments and ****kwargs** allows you to pass a varying number of keywords. The importance of both is the use of the asterisk(s) (* or **), the names can be anything you want them to be.

For some more tips and examples, check out the following websites:

- freeCodeCamp - <https://www.freecodecamp.org/news/args-and-kwargs-in-python/>
- DZone - <https://dzone.com/articles/adding-functionality-to-legacy-code>
- Codecademy - <https://www.codecademy.com/resources/blog/how-to-work-with-code-written-by-someone-else/>

3.8. Syntactic, Runtime and Semantic Errors – Handling Errors in Python

There are three very important different types of errors that can occur when writing a program and it is useful to know their differences and how to spot them early on:

1. **Syntax errors** occur when Python is translating the source code into binary form and indicates that some part of the syntax is wrong, e.g., indentation, missing a colon at the end of a def statement or a missing parenthesis. Similar to natural languages, when we use incorrect syntax in Python, we will get an error saying: invalid syntax.

Syntax errors are easier to spot, here are some things to look out for:

- Using a Python keyword for a variable name
- Missing a colon (:) at the end of **for**, **while**, **if** and **def** statements



- Indentation within loops and conditionals
- Matching quotation marks for strings i.e., both double or single
- Check that you have not missed any quotation marks when writing strings
- Using one equal sign instead of two in a conditional statement
- An unclosed bracket in any lines of your code i.e.,),], }

If you cannot find the syntax error, create a new file and add your code line by line from the beginning.

2. **Runtime errors** occur when the following apply: 1) the program does not do anything, 2) the program enters into an infinite loop or recursion, 3) you get an exception error or 4) you have added too many print statements.

Let's see some ways that you can overcome or spot the root for any those instances:

- When your program does not do anything, make sure that you have called it to execute in the interactive console
 - If you enter an infinite loop, stop the program and add *print* statements where you think the problem might be and try to use hashtags (#) in front of pieces of your code to see what happens
 - Exception errors fall into 5 main categories:
 - i. **NameError** where you are trying to use a variable that does not exist, i.e., local variables;
 - ii. **TypeError** can occur for multiple reasons from using a value incorrectly, mismatch between items in string format and converted items or wrong number of arguments;
 - iii. **KeyError** can occur when trying to access an item in a dictionary using a key that does not exist;
 - iv. **AttributeError** can occur when trying to access an attribute that does not exist;
 - v. **IndexError** where there is mismatch between the index number of a list, string or tuple and its length.
3. **Semantic errors** are usually the most difficult errors to figure out since the program is running but does not produce the wanted outcome. You might have thought that the order you put your statements in makes sense, however, Python might behave in unexpected ways.

When running into semantic errors, it is helpful to do the following:

- Break down the code into smaller parts to figure out how the program is behaving at each step
- Review your thoughts or notes on the logic behind each line of code
- Use print statements to see what the program is doing



- If you have long expression, use temporary variables to check the types of variables
- Assign a variable to your expression before a return statement
- If you cannot figure out the error, take a short break or ask for help

3.9. Practice

One of the most important aspects of coding is practice. The more you practice, the better you will become. Hopefully by the end of this module, you have a solid understanding of the logic behind writing code, the different data types and their usage, as well as how to build small programs using functions.

We encourage you to practice as much as possible in order to use coding to create your own individual projects and improve your career prospects and success.

Ready to start practising? Happy Coding!

Here, you can find a list of different websites that you can use to practice your coding skills further:

- ⇒ **W3Schools** - https://www.w3schools.com/python/python_exercises.asp
- ⇒ **GeeksforGeeks** - <https://www.geeksforgeeks.org/python-exercises-practice-questions-and-solutions/>
- ⇒ **PYnative** - <https://pynative.com/python-exercises-with-solutions/>

References:

Downey, A. Elkner, J. & Meyers, C. (2008). How to Think Like a Computer Scientist: *Learning with Python*. Green Tea Press: Wellesley, Massachusetts.

GeeksforGeeks (2021). *Top 10 Python IDE and Code Editors in 2020*.

<https://www.geeksforgeeks.org/top-10-python-ide-and-code-editors-in-2020/>

Karpinski, Z., Biagi, F., & Di Pietro, G. (2021). Computational Thinking, Socioeconomic Gaps, and Policy Implications. IEA Compass: Briefs in Education. 12. *International Association for the Evaluation of Educational Achievement*.

O' Dea, B. (2021). *Python named most in-demand coding language for 2022*.

<https://www.siliconrepublic.com/careers/python-most-in-demand-coding-language-2022>

Programiz. *How to Get Started with Python?* <https://www.programiz.com/python-programming/first-program>

Real Python. *Python args and kwargs: Demystified*. <https://realpython.com/python-kwargs-and-args/>

W3Schools. *Python Tutorial*. <https://www.w3schools.com/python/>



PART B: The CodER Microcontrollers Module (10 hours)

Description:

This part provides an introduction to microcontrollers, their usage and their application based on real-life examples. The first subsection explains the components of a microcontroller and its different types to familiarise learners with this concept. Then, it moves on to the use of the Arduino software and how it connects to Arduino microcontrollers in a step-by-step approach.

Workload:

10 hours

Learning outcomes:

By the end of this course, learners will be able to:

- ⇒ Recognise what a microcontroller is and be able to identify the different types of microcontrollers
- ⇒ Differentiate between Analog and Digital Input/Outputs (I/O)
- ⇒ Use basic syntax of Arduino IDE
- ⇒ Execute different examples of Arduino IDE and microcontrollers

Required material and resources:

- ⇒ Computer or laptop
- ⇒ Internet Access
- ⇒ Arduino Uno
- ⇒ Arduino IDE

Practicalities:

This part of the module is designed to cover 10 hours of learning. Each section of the module has a designated time, but the learner or educator/trainer is free to decide how much to spend on each subtopic depending on prior knowledge and engagement in similar topics. The content is based on progressive development and is used to provide basic knowledge and skills to beginners.

Subsections:

1. Introduction to Microcontrollers
2. Fundamentals of programming with Arduino
3. Applications of Arduino



1. Introduction to Microcontrollers

- ⇒ **Number of participants:** 1-10 per facilitator
- ⇒ **Duration:** 1.5h
- ⇒ **Teaching methods:** Presentation, Guided instruction, Experiential learning
- ⇒ **Required materials:** Presentation, Arduino Boards and stable internet connection

1.1. What is a microcontroller

Before we explain what a microcontroller is, let's consider the following questions:

- Were you ever curious about how gadgets work? What is the logic behind them?
- Have you ever wanted to know how systems that control elevators work or electronic toys?
- Or even create your own robot or electronic signals for a model railroad?
- Have you ever wondered how weather data is captured and analysed?

Are you curious? Well, let's get started then!

Microcontrollers can facilitate a better understanding of these electronic processes through hands-on activities.

A microcontroller is a compact integrated circuit responsible for executing a specific function in a device. It interprets the data it receives from its input and output (I/O) peripherals through its central processor.

Microcontrollers can be found in a variety of systems and devices. Some applications of microcontrollers can be found in cameras, motor controls, door locks, fire or smoke alarms, or sensors of temperature, light, and colour.

Consider the example of a car with many microcontrollers to control individual systems such as light sensors, anti-lock braking systems, power windows, or brake controls. A vehicle can have as many as 50 microcontrollers responsible for specific operations that either communicate with each other or with other more complex systems to perform appropriate actions.

A microcontroller is essentially a small chip, which is comprised of the following elements:

1. **The processor (CPU):** A processor or central processing unit (CPU) is used to process and execute various instructions that direct a microcontroller's function. It reads and performs basic arithmetic, logic and I/O operations. It is also responsible for transferring data to communicate commands to other components within a larger embedded system.
2. **Memory:** The main function of a microcontroller's memory is to store data in order to send it to the processor and respond to its predetermined programming function.

A microcontroller has two main types of memory:



- a. Program memory is the one that stores long-term information about the operations that the microcontroller was programmed to execute. This type of memory does not need a power source to work and stores information for a long period of time.
 - b. Data memory or Random Access Memory (RAM) is used to store temporary data during the execution of the program. This type of memory holds data as long as the device is connected to a power source.
3. Input/Output (I/O) peripherals: The I/O peripherals are responsible for microcontrollers' communication with other components. The input ports receive information and send it for further processing to the processor as binary data. Based on the processor's commands, the output ports execute the necessary tasks.

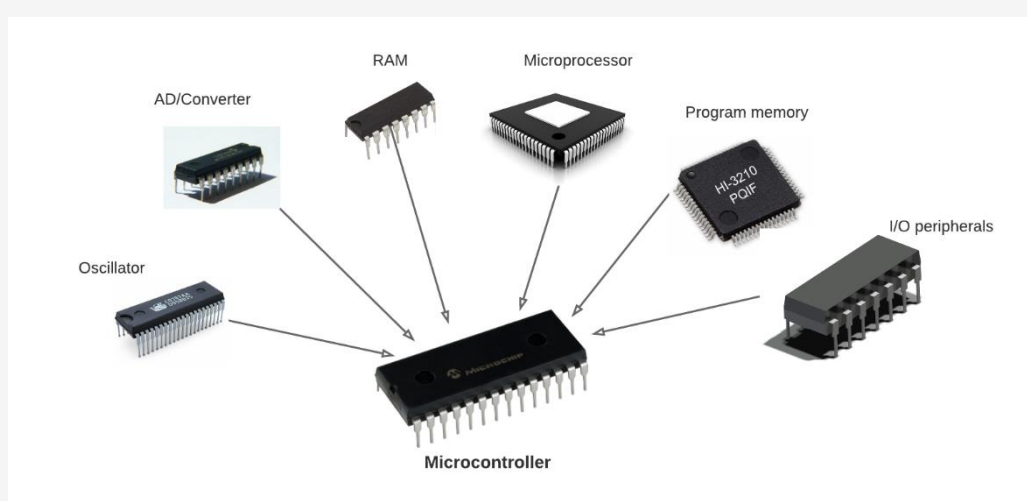


Figure 16 – Elements of a microcontroller.

Source: <https://www.circuitbasics.com/introduction-to-microcontrolleres/>

There are many examples of microcontrollers available in the market. Arduino, Scratch, Microbit and Raspberry PI are amongst the most popular.

Here are their official websites for you to check them out in your own time:

- Arduino: <https://www.arduino.cc/en/software>
- Scratch: <https://scratch.mit.edu/>
- Microbit: <https://microbit.org/>
- Raspberry PI: <https://www.raspberrypi.org/>

In this module, we will focus on Arduino microcontrollers.

1.2. What is Arduino and its different types

What is Arduino

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a

piece of software or IDE (Integrated Development Environment) that runs on a computer, used to write and upload computer code to the physical circuit board, which is a great platform for prototyping projects and creations (Green Steam Incubator, 2019).

Arduino boards hold the key to understanding electronic processes through hands-on activities. This system was created by Massimo Banzi and David Cuartielles in 2005. It offers an alternative to the otherwise expensive microcontrollers and allows you to build interactive projects such as GPS tracking systems, light sensors, remote-controlled robots, and electronic games.

The main components of Arduino are:

1. **Software:** The Arduino IDE is responsible for writing programs used to communicate with your hardware. Board functions are controlled based on the instructions sent to the microcontroller via Arduino IDE.
2. **Hardware:** Arduino boards come in various types and they can read analog or digital input signals from different sensors to produce an output. Some examples of outputs include activating a motor, turning LED on/off or locking/unlocking a door.
3. **Programming language:** The programming language used in Arduino is a simplified version of C++.

Types of Arduinos

There are several kinds of Arduino boards available depending on the microcontroller used. The differences are mainly presented in the following features:

- number of inputs and outputs (i.e., sensors, LEDs, buttons on a board)
- speed
- operating voltage
- form factor etc.

Some boards are solely designed to be embedded and do not offer a programming interface. Others can be directly powered from a 3.7V battery, while others need at least 5V to operate. Whatever their differences in these features, they can still be programmed through the Arduino IDE.

A few examples of the different types of Arduino boards can be seen in the picture below.





Figure 17 – Different types of Arduino boards.

Source: <https://techatronic.com/types-of-arduino-boards-arduino-uno-mega-mini-specification/>

Among the most popular Arduino boards is the Arduino Uno. Even though it was not the first board to be released in the market, it still remains one of the most actively and widely documented boards.

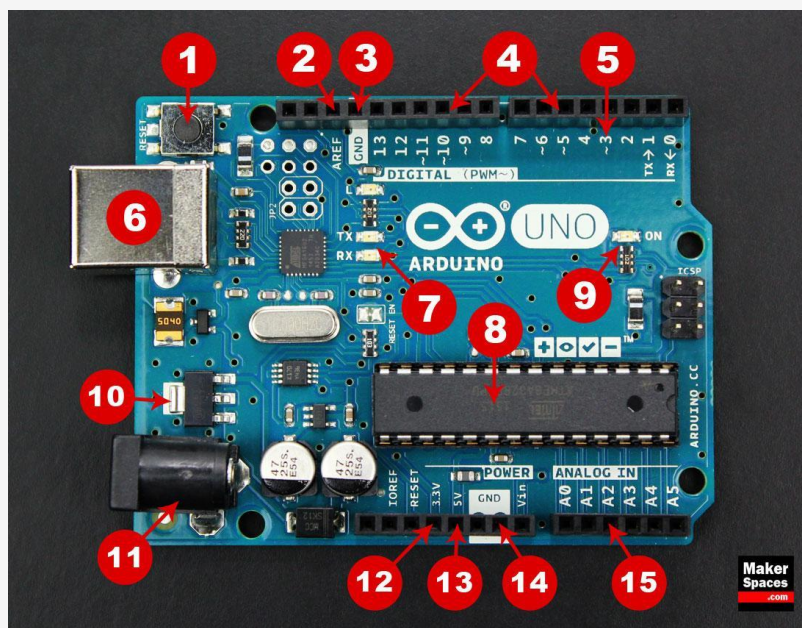


Figure 18 – Arduino UNO.

Source: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>

1. Reset Button – Restarts any code that was loaded to the Arduino board
2. AREF – Stands for "Analog Reference" and sets an external reference voltage
3. Ground Pin (GND) – Closes the electrical circuit and provides a common reference throughout

4. Digital Input/Output – Pins 0-13 can be used for digital input or output
5. PWM – The pins marked with the (~) symbol can simulate analog output
6. USB Connection – Powers your Arduino and uploads sketches
7. TX/RX – Transmits and receives data indication from LEDs
8. ATmega Microcontroller – Stores the programs (brain of the Arduino)
9. Power LED Indicator – Lights up when the board is plugged in a power source
10. Voltage Regulator – Controls the amount of voltage going in the Arduino board
11. DC Power Barrel Jack – Powers your Arduino with a power supply
12. 3.3V Pin – Supplies 3.3 volts of power to your projects
13. 5V Pin – Supplies 5 volts of power to your projects
14. Ground Pins – Close the electrical circuit and provide a common reference throughout
15. Analog Pins – Read the signal from an analog sensor and convert it to digital

Arduino Uno boards need to be connected to a power source to work. There are various ways to connect the board to a power source such as directly through a computer via USB or in the case of mobile projects via a 9V battery pack. The last method requires the use of a 9V AC power supply to work.



Figure 19 – Power sources of Arduinos.

Source: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>

Another important component of the Arduino board is the breadboard, also called *Solderless Breadboard*. It is used in the prototyping phase to assess the circuit's functionality and allows for the temporary creation and experimentation of different circuit designs. The tie points (holes) of the plastic housing contain metal clips connected to each other by strips of conductive material. However, the breadboard is not powered on its own and needs to be connected to the Arduino board via jumper wires. Also, these wires are used to form the circuit by connecting resistors, switches and other components together.

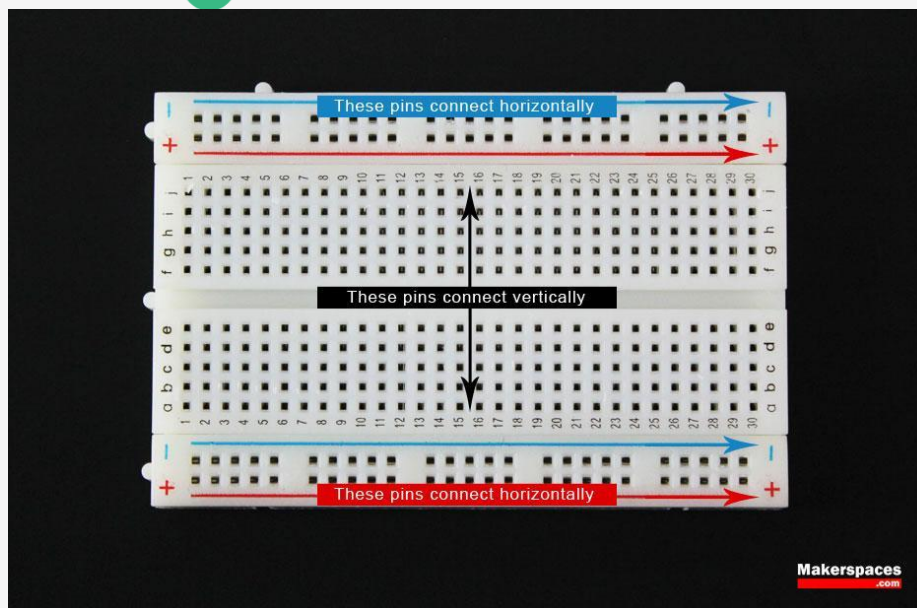


Figure 20 – Breadboard.

Source: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>

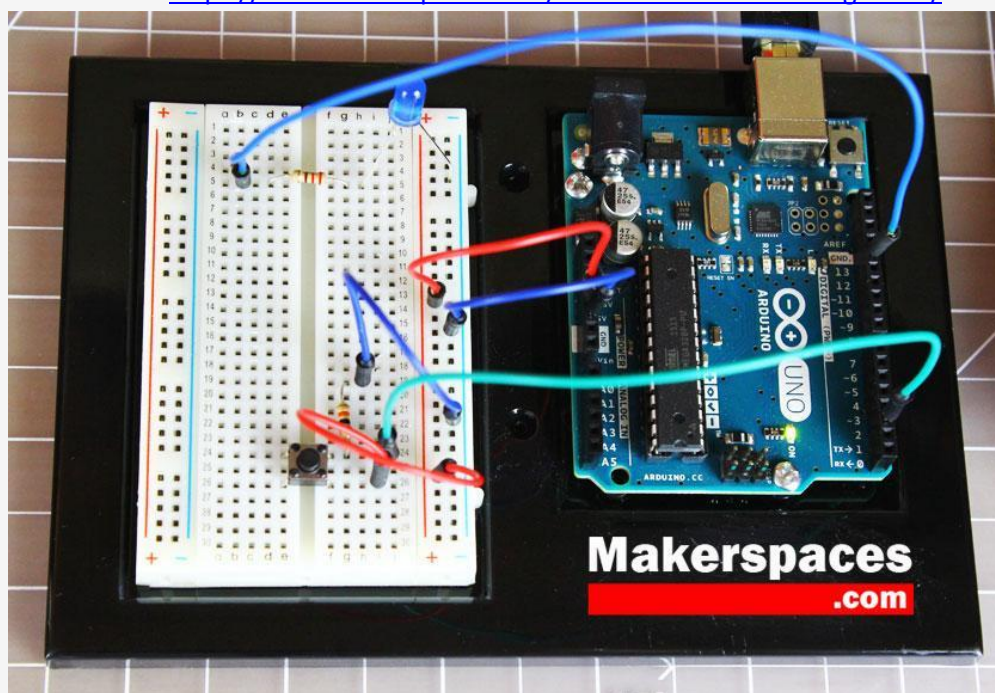


Figure 21 – Completed Arduino circuit.

Source: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>

1.3. Concepts: Input, Output, Analog, Digital

Inputs and Outputs (I/O) within a programming context

Input and Outputs represent the interactions between a robot/machine and the real world. In simpler words, inputs are data the robot/machine receives and outputs are the results we can see.

Inputs are usually transmitted by sensors such as switches, potentiometers or cameras, whereas outputs refer to immediate actions initiated by motors, such as the LED lighting up or the alarm going off.

Consider the following example: The keys on a keyboard represent inputs. While typing, the computer receives information that is ordered. However, this process is not visible to us. The computer or machine takes the input (i.e., whatever is typed) and outputs it on the screen.

In the Arduino context: A program can take a button as input, which will light up a LED light if it is activated. The information processed by the button is not visible to you, but the led light that turns on or off is proof of the output.

There are two types of Inputs/Outputs: Analog and Digital I/Os.

What is the difference between Analog and Digital I/Os

There are two ways to differentiate an Analog signal from a Digital signal:

1. By the sensor type
 2. By the processing method (i.e., time and resolution)
1. Analog vs. Digital Sensors: A LED can either be ON/OFF or have variable intensity, such as ON with low intensity or ON with high intensity.
 - ⇒ If the sensor is a switch placed on the LED, it will control whether the LED will be ON or OFF and will not detect anything else. This is a **digital input**.
 - ⇒ If the sensor is an LDR (light dependent resistor), it will detect the light and will convert it into an analog value that lies between 0-255. We can detect whether a LED is at 0% (off) or at 100% (full brightness) and also read many other values in between (e.g., 20, 23%, 86%). This is an **analog input**.
 2. Analog vs. Digital Processing: To determine if it is Analog or Digital, simply check their processing based on:
 - ⇒ Time: Analog is continuously processing information and whenever input is modified, the output is instantly changed according to the input. The update is immediate. For Digital, processing will have a short delay until the system registers the change. This delay is established by the "sampling frequency" and controlled by a clock.
 - ⇒ Resolution: Analog processing has infinite resolutions because it never stops processing and has many different values. In contrast, Digital resolution works with binary numbers, which means that it only processes two values and signals.



Examples with LED light:

- Switch it on/off with analog processing: when the dimmer is activated, the circuit will immediately change the LED's intensity accordingly.
- Switch it on/off with digital processing: with a set sampling frequency, every 5 seconds, the system will read the switch and indicate if the light is on or off, no matter how intense and bright the light is. When you change the dimmer within these 5 seconds, the LED's light will not change.

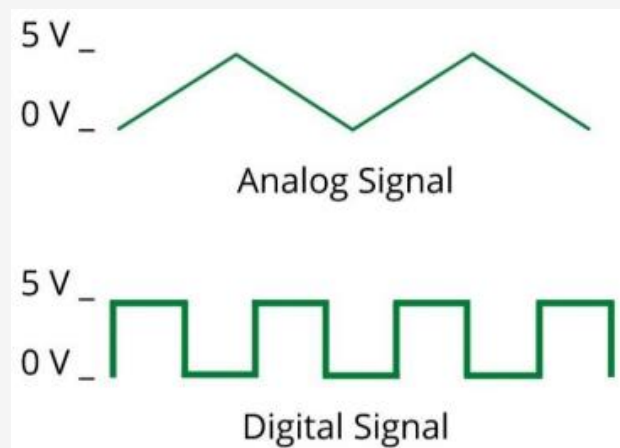


Figure 22 – Analog vs. Digital Signal

Summary

In the real world, analog signals are the most commonly used. However, digital signals and processing are recommended when it comes to robotics. The reasons are: 1) digital processing is cheaper and provides more flexibility compared to analog processing; 2) programs work in a binary form similar to digital signals; and 3) analog signals are more likely to be affected by electrical circuit noise.

Based on these reasons, analog signals are converted into digital signals a lot of the time. It should be noted that all analog signals, both inputs, and outputs, can be converted into digital signals, but not the other way around.

2. Fundamentals of Programming with Arduino IDE

- ⇒ **Number of participants:** 1-10 per facilitator
- ⇒ **Duration:** 3h
- ⇒ **Teaching methods:** Presentation, Guided instruction, Experiential learning
- ⇒ **Required materials:** Presentation, Arduino IDE and Boards, computer (1 per participant) and Stable internet connection

2.1. Setting up Arduino IDE and basic commands

In this section, we will learn how to download and run the Arduino IDE for the first time. We will also go through some basic functions of Arduino IDE in order to get familiarised with the program.

How to download Arduino IDE:

The following instructions are provided to download and set up the Arduino IDE:

1. Go to <https://www.arduino.cc/>
2. Click on Software. It will automatically open the page shown below in Figure 7.

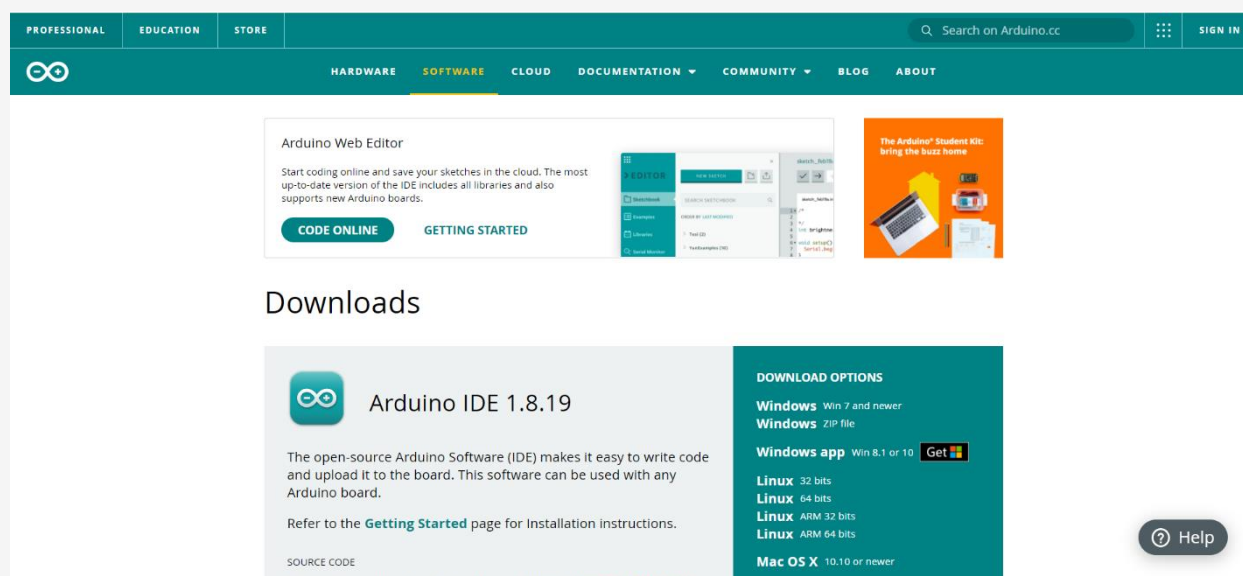


Figure 23 – Downloading Arduino IDE

3. Click on the *download* option that suits your operating system (see Figure 7). If you have doubts, click on “Getting Started” to read more information about installing the software for your computer.

Arduino IDE is used to create, open and change sketches. The board is defined by the sketches we write on the computer through the Arduino IDE. You can either use the buttons shown at the top of the IDE or the menu items.

Once the download is done, the page shown below will open automatically:

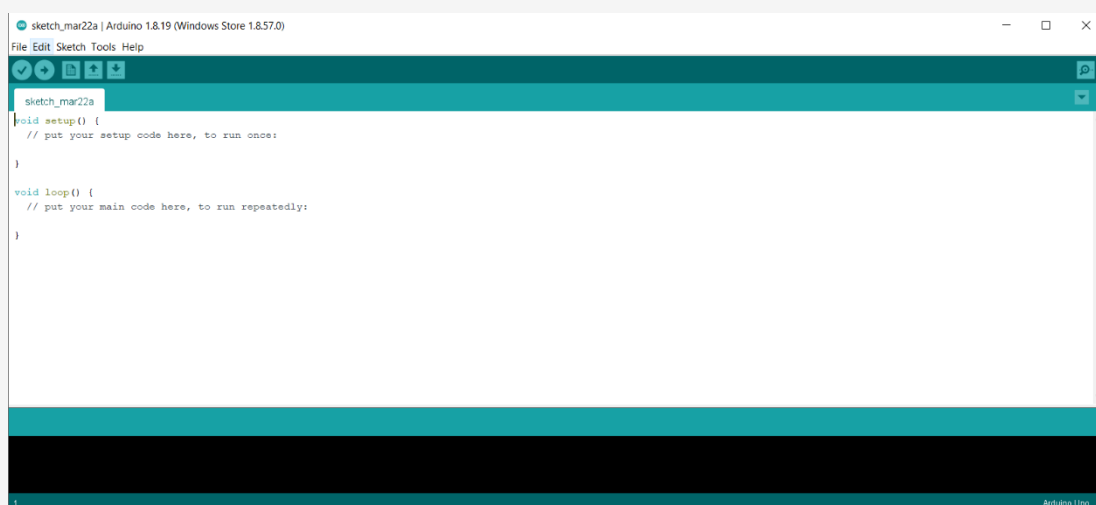


Figure 24 – Opening Arduino IDE for the first time

How to change the language:

1. Click on *FILE* and select *PREFERENCES*.
2. Next to the *Editor Language*, a dropdown menu of currently supported languages is shown.
3. Select your preferred language from the menu.
4. Restart the software to use the selected language.

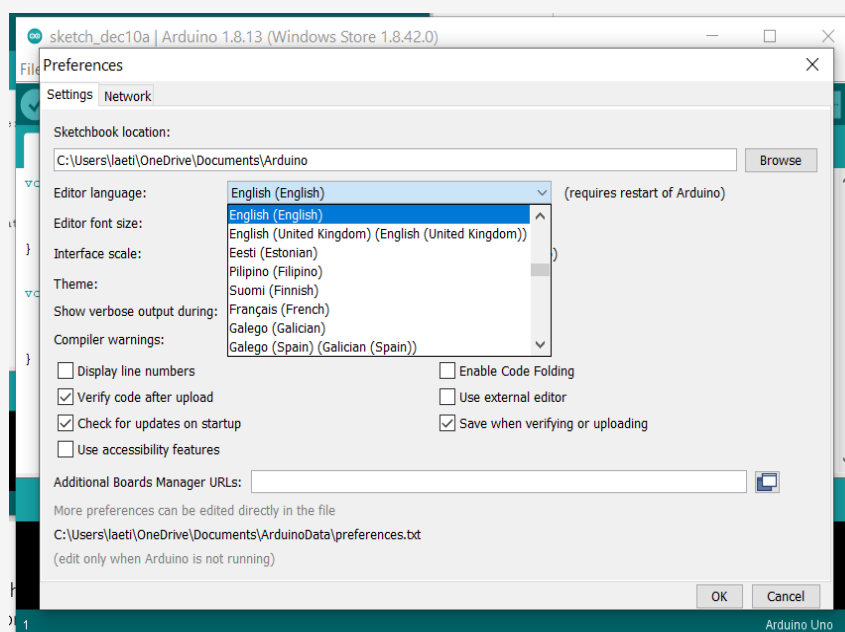


Figure 25 – Changing language settings in Arduino IDE

How to create a new project:

File -> New

It is also possible to use already developed examples to have some inspiration or to reproduce them:

File -> Examples

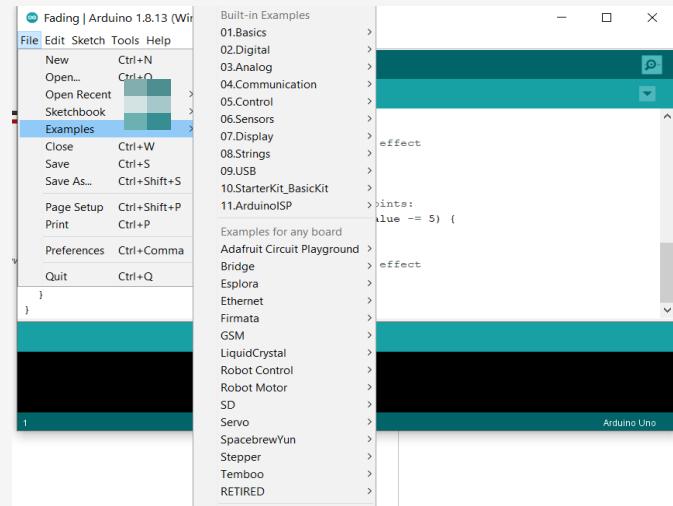


Figure 26 – Creating a new project in Arduino IDE

How to verify a project:

Click on the left below the Tab *FILE*. It will turn orange while it is compiling information. Wait until it comes back to its initial colour.

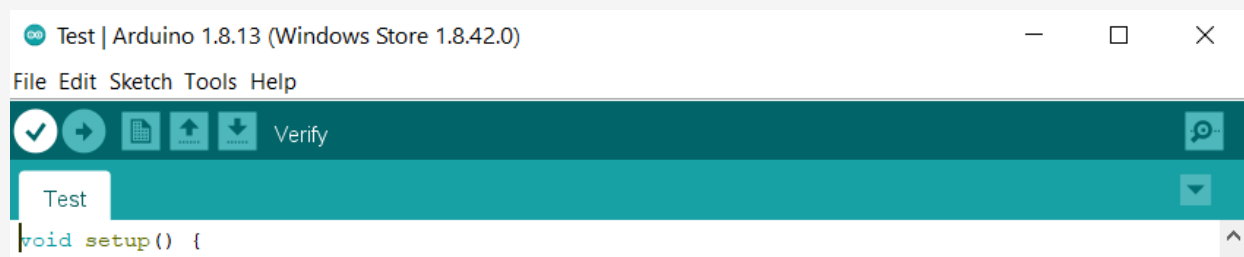


Figure 27 – Verifying a project in Arduino IDE

How to upload a program:

The Arduino board should be plugged into your computer with a USB cable to upload the program. To upload the program, you should click on the horizontal arrow:

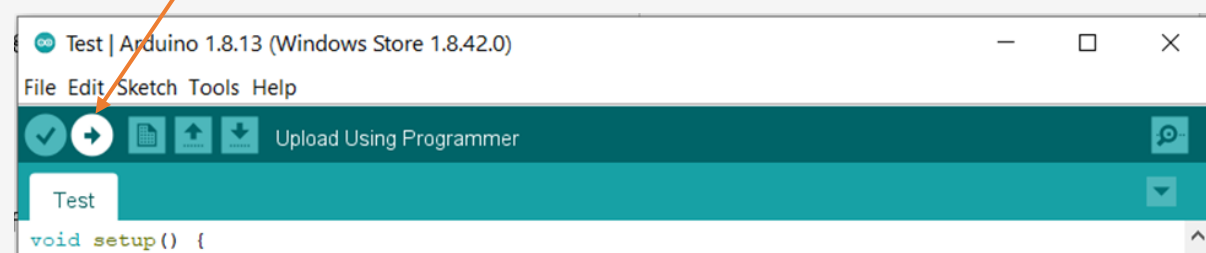


Figure 28 – Uploading a program in Arduino IDE

When the program is uploaded, the arrow will return to its initial colour.

If you have plugged and set up your Arduino board according to what you have programmed, you will be able to watch your program in action.

2.2. Programming in Arduino and uploading programs to the board

How to program in Arduino:

After understanding the hardware of the Arduino UNO board and downloading the Arduino software, we are ready to start programming.

Arduino programs can be divided into three main parts:

1. Structure,
2. Values (variables and constants) and
3. Functions.

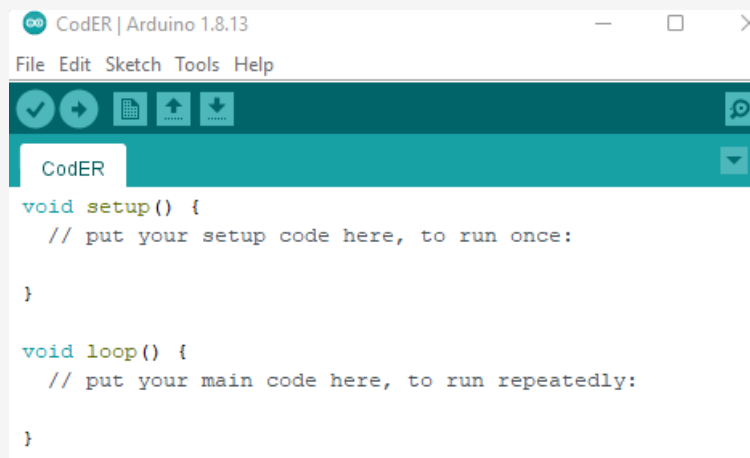
In this session, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Arduino – Programme Structure

Let us start with the Structure. Software structure consists of two main functions:

Setup() function

Loop() function



```

CodER | Arduino 1.8.13
File Edit Sketch Tools Help
CodER
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

```

Figure 29 - Arduino – Programme Structure

The setup() function is called when a sketch starts. We use it to initialise the variables, pin modes, start using libraries, etc. The setup function will only run once after each power-up or reset of the Arduino board. After creating a setup() function, which initialises and sets the initial values, the loop() function does precisely what its name suggests and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Data Types

The Arduino environment is similar to C++ with library support and built-in assumptions about the target environment to simplify the coding process. Below is a list of some data types commonly used in Arduino:

- boolean (8 bit): true/false
- byte (8 bit): unsigned number from 0-255
- char (8 bit): signed number from -128 to 127.
- word (16 bit): unsigned number from 0-65535
- int (16 bit): signed number from -32768 to 32767.
- unsigned long (32 bit) - unsigned number from 0-4,294,967,295.

Arduino – Variables

Variables in the C programming language, which Arduino uses, have a property called scope. A scope is a region of the program, and there are three places where variables can be declared. They are:

- Inside a function or a block, which is called local variables.
- In the definition of function parameters, which are called formal parameters.
- Outside of all functions, which are called global variables.

Example of variables:

This example creates an integer called 'countUp,' initially set as the number zero. The variable goes up by one in each loop.

```
int countUp = 0;           //creates a variable integer called 'countUp'

void setup() {
  Serial.begin(9600);     // use the serial port to print the number
}

void loop() {
  countUp++;             //Adds 1 to the countUp int on every loop
  Serial.println(countUp); // prints out the current state of countUp
  delay(1000);
}
```

Figure 30 – Arduino Variables Example.

Source: <https://www.arduino.cc/reference/en/language/variables/data-types/int/>

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Comparison Operators
- Boolean Operators
- Bitwise Operators
- Compound Operators

More details about each type of operator can be retrieved from here:

https://www.tutorialspoint.com/arduino/arduino_operators.htm

2.3. Blinking Led with Arduino

In this part, we create and upload a simple sketch that will cause a LED to blink repeatedly by turning it on and off for 1-second intervals.

Steps:

- Connect the Arduino to the computer with the USB cable.
- Open the IDE
- Choose *Tools4Serial* Port. Ensure the USB port is selected and that the Arduino board is properly connected.
- Connect a LED to the Arduino's digital pin 13 (as shown in the figure below). A digital pin can either detect an electrical signal or generate one command. In this small project, we will generate an electrical signal that will light the LED.

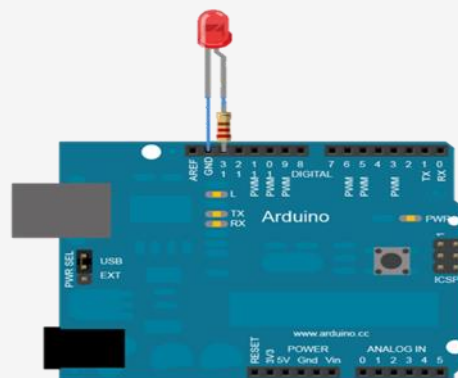


Figure 31 – Arduino Digital Pin in Blinking Led Example

Source: <https://www.instructables.com/How-to-Blink-LED-Using-Arduino/>

- Enter the following into your sketch between the braces { and }, under the void setup():

```
pinMode(13, OUTPUT); // set digital pin 13 to output
```

The number 13 in the listing represents the digital pin you're addressing. You are setting this pin to OUTPUT, which will generate (output) an electrical signal. If you wanted it to detect an incoming electrical signal, you would use INPUT instead. Notice that the function `pinMode()` ends with a semicolon (;). Every function in your Arduino sketches will end with a semicolon.

- Save your sketch again to ensure that you don't lose any of your work by choosing File > Save As.
- Enter a short name for your sketch, and then click OK.

Remember that our goal is to make the LED blink repeatedly. We will create a loop function to tell the Arduino to execute an instruction repeatedly until the power is shut off or someone presses the RESET button.

- Enter the code shown in boldface after the `void setup()` section in the following listing to create an empty loop function.
- End this new section with another brace (`}`), and then save your sketch again.

```
void setup()
{
  pinMode(13, OUTPUT); // set digital pin 13 to output
}
void loop()
{
```

// place your main loop code here:

```
}
```

- Next, enter the actual functions into the `void loop()` for the Arduino to execute.
- Enter the following between the loop function's braces and then, click *Verify* to make sure that you have entered everything correctly:

```
digitalWrite(13, HIGH); // turn on digital pin 13
delay(1000); // pause for one second
digitalWrite(13, LOW); // turn off digital pin 13
delay(1000); // pause for one second
```

The **`digitalWrite()`** function controls the voltage that is output from a digital pin: in this case, pin 13 to the LED. By setting the second parameter of this function to **HIGH**, a "high" digital voltage is output; then current will flow from the pin, and the LED will turn on.

With the LED turned on, the light pauses for 1 second with `delay(1000)`. The **`delay()`** function causes the sketch to do nothing for a period of time— in this case, 1,000 milliseconds or 1 second.

- Next, we turn off the voltage to the LED with `digitalWrite(13, LOW);`



- Finally, we pause again for 1 second while the LED is off, with `delay(1000);`

The completed sketch should look like this:

```
void setup()
{
  pinMode(13, OUTPUT); // set digital pin 13 to output
}
void loop()
{
  digitalWrite(13, HIGH); // turn on digital pin 13
  delay(1000); // pause for one second
  digitalWrite(13, LOW); // turn off digital pin 13
  delay(1000); // pause for one second
}
```

- Save your sketch.
- Verify your sketch, to ensure that it has been written correctly so that the Arduino can understand.
- Once the sketch has been verified, a note should appear in the message window, as shown below:

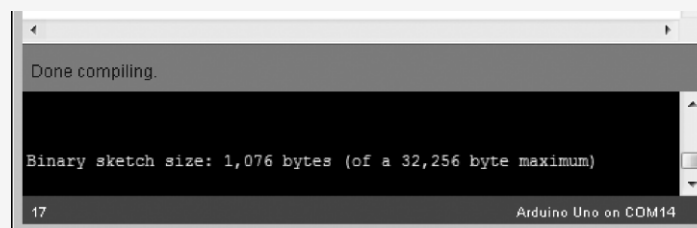


Figure 32 – Verification of Sketch in the Blinking Led Example

- Ensure that your Arduino board is connected and click Upload in the IDE. The IDE may verify your sketch again and upload it to your Arduino board.

The TX/RX LEDs on your board should blink during this process, indicating your program functions properly.

3. Applications

- ⇒ **Number of participants:** 1-10 per facilitator
- ⇒ **Duration:** 5.5h
- ⇒ **Teaching methods:** Presentation, Guided instruction, Experiential learning
- ⇒ **Required materials:** Presentation, Arduino IDE and Boards, computer (1 per participant), Stable internet connection and relevant materials depending on the project (e.g., paintbot)

3.1. What is robotics?

Robotics is a field where science, engineering and technology intersect. The goal is to produce machines, called robots, capable of substituting, replicating or assisting human actions. Originally, robots were built to handle monotonous tasks, especially in industry. But since their creation, they have expanded beyond their initial uses. Nowadays, we can find an enormous variety of robots that can perform a wide range of tasks like fighting fires, cleaning homes and assisting doctors with incredibly intricate surgeries. Each robot includes a differing level of autonomy, starting from human-controlled bots that perform tasks that a human has full control over to fully-autonomous bots that perform tasks with no external influences.

The world of robotics is clearly expanding but still, we can find some consistent characteristics:

1. “Robots consist of some sort of mechanical construction. The mechanical aspect of a robot helps it complete tasks in the environment for which it’s designed” (Built In, 2022, §3).
2. “Robots need electrical components that control and power the machinery” (Built In, 2022,§3).
3. “Robots contain at least some level of computer programming. Without a set of code telling it what to do, a robot would just be another piece of simple machinery. Inserting a program into a robot gives it the ability to know when and how to carry out a task” (Built In, 2022, §3).

3.2 Types of robots

In today’s world, we can find a range of robots used for various applications. As technology advances, robots become more and more important in our everyday lives where they play an important role. There are many types of robots and they vary a lot from one to another, from mini robots to huge industrial robots, from concierge robots to entertainment robots.

We can classify robots in many different ways, considering their types and applications. Every robot has its own unique features and it can vary a lot in size, shape and capabilities. Nonetheless, many robots share a spread of features. Here are the 13 categories we can use to classify robots.



Industrial robots

In the robot industry, we can find mainly six types of robots: articulated robots, Cartesian robots, SCARA robots, cylindrical robots, delta robots and polar robots.

- **Articulated Robot:** It resembles a human arm in its mechanical configuration. The arm is connected to the base with a twisting joint. The number of rotary joints connecting the links in the arm can range from two joints to ten joints and each joint provides an additional degree of freedom (Analytics Insight, 2021).



Figure 33 – Articulated robot

Source: <https://diy-robotics.com/article/articulated-robots/>

- **Cartesian:** Also called rectilinear or gantry robots, Cartesian robots have three linear joints that use the Cartesian coordinate system (X, Y, and Z). They also may have an attached wrist to allow for rotational movement. The three prismatic joints deliver a linear motion along the axis (Analytics Insight, 2021).

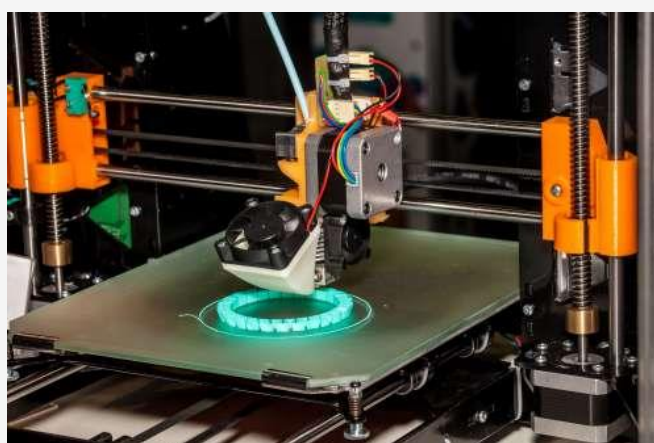


Figure 34 – Cartesian robot

Source: <https://diy-robotics.com/article/what-you-should-know-about-cartesian-robot/>

- **SCARA:** Selective Compliance Assembly/Articulated Robot or Arm (SCARA) is more commonly used for assembly purposes all over the world due to its easy and unobstructed mounting (Analytics Insight, 2021).



Figure 35 – SCARA Arm

Source: <https://diy-robotics.com/article/scara-robots/>

- **Cylindrical:** These robots are generally used for assembly purposes, spot welding and machine die castings. They have a minimum of one rotary joint at the base and at least one prismatic joint to connect the links. The rotary joint uses a rotational motion along the joint axis, while the prismatic joint moves in a linear motion (Analytics Insight, 2021).

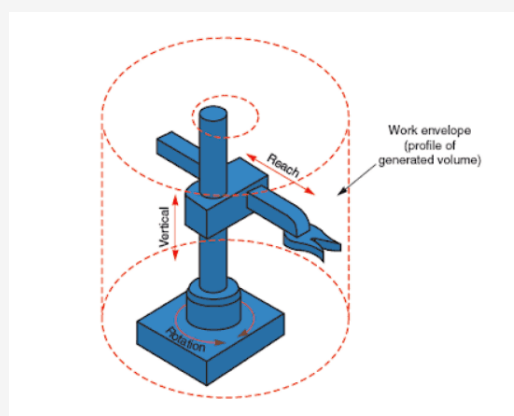


Figure 36 – Cylindrical Work Envelop

Source: <https://learnmech.com/cylindrical-robot-diagram-construction-applications/>

- **Delta Robots:** Also referred to as “spider robots,” they use three base-mounted motors to actuate control arms that position the wrist. Basic delta robots are 3-axis units but 4- and 6-axis models are also available (Analytics Insight, 2021).



Figure 37 – Delta robots

Source: <https://diy-robotics.com/article/top-six-types-industrial-robots-2020/>

- **Polar:** Also known as spherical robots, in this configuration the arm is connected to the base with a twisting joint and a combination of two rotary joints and one linear joint. The axes form a polar coordinate system and create a spherical-shaped work envelope (Analytics Insight, 2021).

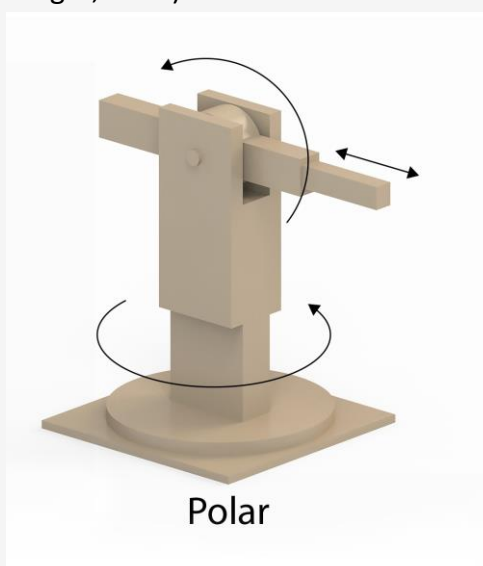


Figure 38 – Polar robots

Source: <https://www.howtorobot.com/expert-insight/industrial-robot-types-and-their-different-uses>

Apart from industrial robots, there are limitless kinds of robots in every field like education, security, space science, medicine and more. Here you can find some examples of those robots but remember that you can find many more.

Domestic robots

Also called consumer robots are robots you can buy and use just for fun or to help you with tasks and chores. These robots are used to perform household tasks that include pool cleaning robots, robot vacuum cleaners, gutter cleaning robots, AI-powered robot assistants, and a

growing variety of robotic toys and kits. This category may include entertainment robots, they are designed to arouse human emotions to entertain us.

Military robots and disaster response robots

Robots are also used in the military or for an emergency response where the situation may be too dangerous for humans. Robot drones, mine detectors, and sensing devices are common on the battlefield but require direct control by humans. Remotely piloted vehicles prevent the loss of human life that would occur if soldiers were on the field instead of being behind controllers (Stanford Edu, 2022). Nonetheless, the use of robots and AI in military operations is quite controversial and can raise many legitimate ethical concerns. Furthermore, these types of robots are able to endure extreme conditions and traverse difficult fields. These robots perform dangerous jobs like searching for survivors in the aftermath of an emergency and helping in other crucial activities at the disaster site (Analytics Insight, 2021).

Medical robots

Robots have had a huge impact on medicine. They started about 35 years ago when the goal was to insert a probe into the brain to obtain a biopsy specimen. “Today, medical robots are well known for their roles in surgery, specifically the use of robots, computers and software to accurately manipulate surgical instruments through one or more small incisions for various surgical procedures. A 3D high-definition magnified view of the surgical field enables the surgeon to operate with high precision and control” (NCBI, 2019). One of the most famous medical robots is the one produced by da Vinci, approved by the FDA in 2000 and it is said to have been used to perform over 6 million surgeries, worldwide (NCBI, 2019).

Service Robots

“The International Organization for Standardization defines a “service robot” as a robot “that performs useful tasks for humans or equipment excluding industrial automation applications” (IFR, 2022). The most common types of service robots are concierge robots and hospitality robots. Guardforce Hong Kong, for instance, has launched a concierge robot that automates visitor registration, controls access points through facial recognition and offers multimedia information, making it perfect for commercial buildings, events and exhibitions. The same company has a hospitality robot for malls that provides a shopping guide, a handshake and chat function and the ability for customers to download coupons (Guardforce, 2022).

Cobots

Collaborative robots or “cobots” work together with humans in a shared environment to perform their tasks. E.g., the Sawyer robot arm helps greenhouse workers pick plants. Mitsubishi robot offers coffee at Café X kiosk in Hong Kong.



Drones

Also called unmanned aerial vehicles (UAV), which is actually an aircraft without a human pilot on board. Drones can be of different sizes and have different levels of autonomy. This type of robot has invaded a wide range of industries around the world and it helps in carrying out many activities such as atmosphere clearing, aerial photography and delivery.

Humanoid robots

This is probably the type of robot that most people think of when they think of a robot. These robots look like and can also mimic human behaviour. They usually perform human-like activities (like running, jumping and carrying objects) and are sometimes designed to look like us, even having human faces and expressions (Built In, 2022).

Educational robots

Educational robotics is a new discipline designed to introduce students to Robotics and Programming in an interactive way from a very early age. Educational robotics provides students with everything they need to easily build and program a robot capable of performing various tasks, the complexity of the discipline is always adapted to the students' age (Iberdrola, 2022).

3.3 Driving a DC motor with a motor shield

Typically, robots and automated devices contain moving parts. Movement is enabled by motors that are programmed to function in a certain way. In this section, we explore the programming of a basic DC motor, a motor that can spin clockwise and counterclockwise. DC motors are generally employed in toys as well as in electrical appliances such as mixers and other kitchen gadgets.

It is possible to run a DC motor with a programmable board (ex. Arduino) using just a few components. However, for simplicity's sake, it is generally agreed that the best solution is to couple the DC motor with a motor shield.

There are two main strategies to wire a DC motor to an Arduino board. The first involves using a Mosfet and a diode, the second one relies on an electronic component called a motor shield.

A motor shield allows us to bypass a number of complicated wirings that are necessary to operate one or more DC motors.

There exist several motorshields. For the following exercise, you will need the Adafruit motor shield V1. Be careful to choose Version 1 and not Version 2 of the Adafruit motor shield, as the programming of one is not compatible with the other.



The Adafruit motor shield V1 contains two L293D chips. For this reason, it can drive up to 4 DC motors simultaneously.

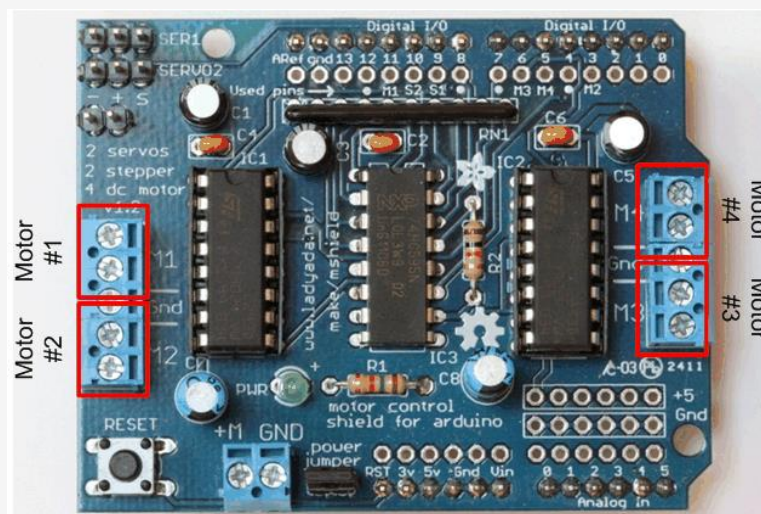


Figure 39 – Adafruit motor shield V1

Source: <https://learn.adafruit.com/adafruit-motor-shield/af-dcmotor-class>

Besides DC motors, it is possible to drive servo motors and stepper motors with the adafruit motor shield.

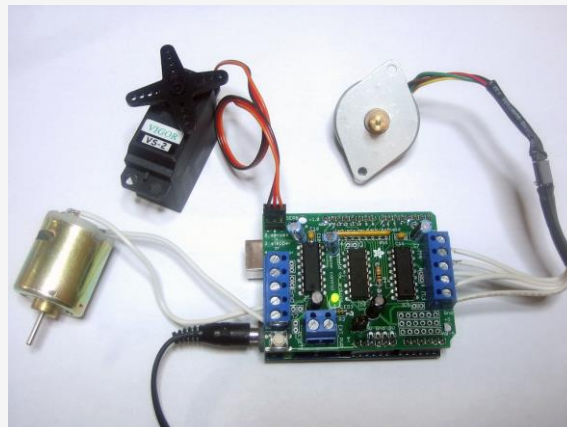


Figure 39 – Adafruit motor shield V1 connected

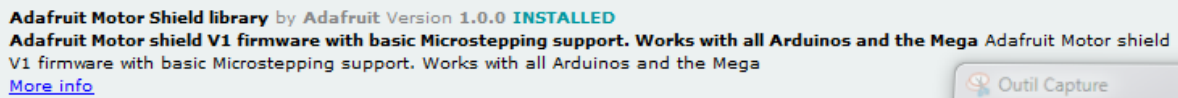
Source: <https://learn.adafruit.com/adafruit-motor-shield/af-dcmotor-class>

You can connect one DC motor to the shield by joining the motor cables to either M1, M2, M3 or M4.

When you're done, you can get started with the coding task.

In order to program the DC motor connected to our motor shield, we first need to install the adafruit motor shield library. You will find it in the library directory by searching for "AF motor".

Go to Sketch > Include Library > Manage Libraries...



Adafruit Motor Shield library by Adafruit Version 1.0.0 **INSTALLED**
 Adafruit Motor shield V1 firmware with basic Microstepping support. Works with all Arduinos and the Mega
[More info](#)

At this stage, you can start programming the motor shield. A simple code to make the DC motor spin is the following:

```
#include <AFMotor.h>

AF_DCMotor motor1(1); // We define a motor attached to M1 on the motor shield

void setup()
{
  motor1.setSpeed(100); // We define the speed with which the motor will spin
}

void loop()

{
  motor1.run(BACKWARD); // the motor will spin either clockwise or counterclockwise,
  depending on the way you connected it to the motor shield. To make it spin the opposite
  direction you need to replace BACKWARD with FORWARD.

  delay(10000); // it will spin for 10 seconds

  motor1.run(RELEASE); // it will stop for 1 second and begin from the top of the loop
  function again

  delay(1000);
}
```

3.4. Build a paintbot using a DC motor and Arduino

It is possible to implement the programming of a DC motor in an artistic setting to achieve something interactive and creative. A paintbot merges arts with electronics and it's a machine that uses paint to create unique artworks.



Figure 40 – Artwork using a paintbot

Source: <https://girlsinstem.eu/>

The PaintBot consists of a turning table, rotating at varying speeds. On the turning table, you can place a sheet of paper or a small canvas. By carefully dropping paint drops above the rotating sheet, you get a unique masterpiece created by you and the PaintBot.



Figure 41 – Components of a paintbot

Source: <https://girlsinstem.eu/>

Create the box

- A. Take a red and black piece of wire of at least 30 cm each and connect it to the motor. (Be sure not to cut the pieces too small as this complicates connecting the motor in the PaintBot to the rest of the electronic circuit. The bigger your box, the longer the wires need to be)
- B. Use the egg carton as a motor stand
 1. Remove the lid of the egg carton, we do not need it.

2. We need one of the tops of the egg carton and its surrounding 4 egg spots. Cut between the 3rd/4th egg spot and the adjacent top.
 3. Cut the tip of the top. It is better to start cutting a small piece and adjust it afterwards rather than cutting too much.
 4. Put the motor from the bottom to the top with the motor shaft upwards and the wires pointing down. The motor should fit tightly in the hole you just cut. You can try to push it carefully or cut a bit more from the top. It is important that the motor fits firmly and the construction is robust as it forms the base of the turning table. To give extra stability and support you can use tape or you can put toothpicks underneath the motor through the egg carton.
- C. Make a little hole in the middle of the cardboard box. Put the wires of the motor through the hole going from the inside of the box to the outside or bottom of the box. Place the egg carton holder on top of it.
 - D. Ensure that the egg carton holder is placed in the middle of the box and firmly attach it to the box. This is very important as it needs to withstand the force of the rotating motor.
 - E. Place a carton disc (round or any other shape) on top of the red (3D printed) motor connection disc. Make sure it is smaller than the box so it can freely rotate.

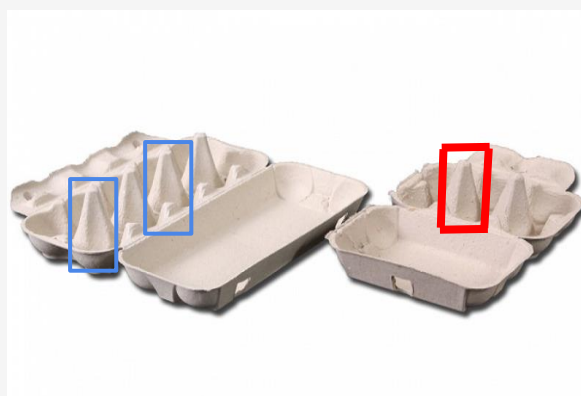


Figure 42 – Step B of the paintbot

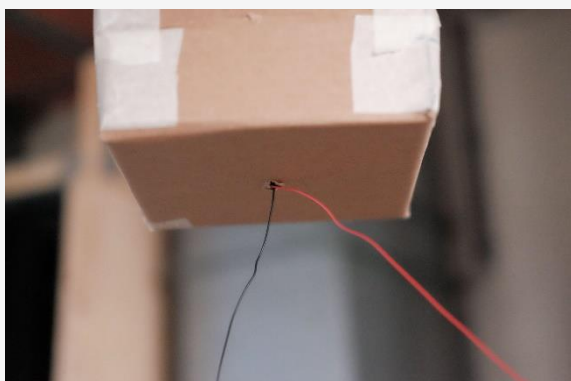


Figure 43 – Step C of the paintbot



Figure 44 – Step D of the paintbot

Connect the electronics

After creating the box, you can simply connect the DC motor that's attached to the motor shield and program it to change speed and direction in order to create your artwork!

3.5 Build an interactive paper toy with a servo motor

A servomotor is any motor-driven system with a feedback element built-in.

If you open up a standard servo motor, you will almost always find three core components: a DC motor, a controller circuit and a potentiometer or similar feedback mechanism. The DC motor is attached to a gearbox and output/drive shaft to increase the speed and torque of the motor. The DC motor drives the output shaft. The controller circuit interprets signals sent by the controller and the potentiometer acts as the feedback for the controller circuit to monitor the position of the output shaft.

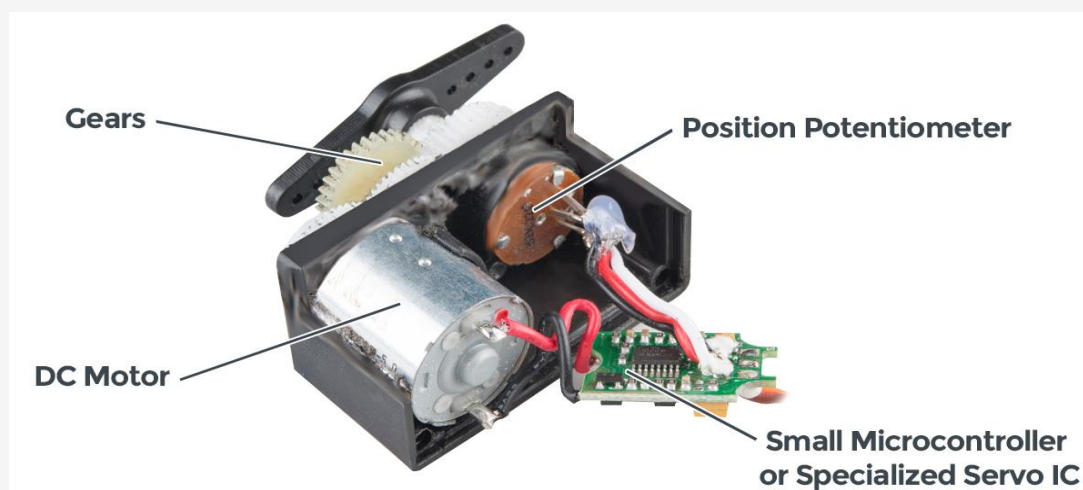


Figure 45 - Servomotor

Source: <https://www.sparkfun.com/servos>

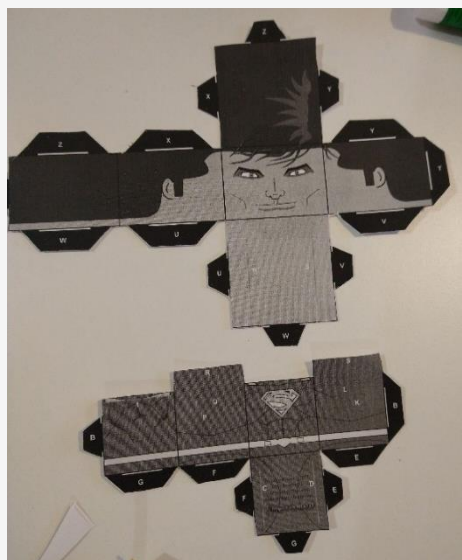
Servo Motors have countless applications. In this section, we illustrate how to use a servo motor to create an interactive paper toy whose head you can turn left or right by acting on a potentiometer.

There will be no programming for this project, however, it is possible to achieve the exact same results by coupling the servo motor with an Arduino board.

Step 1: Create the paper toy

We will start by creating our paper toy. First off, you need to choose a template, then cut out the template from paper or cardboard and finally assemble the paper toy according to the instructions.

Different templates to choose from are available on [Cubeecraft](#).



Step 2: Build the electronic circuit

We start by placing the NE555 chip in the middle of a breadboard, just like in the photo below.

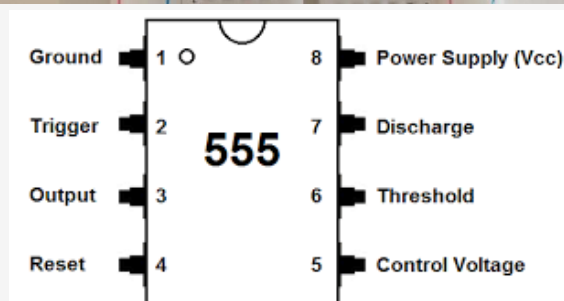
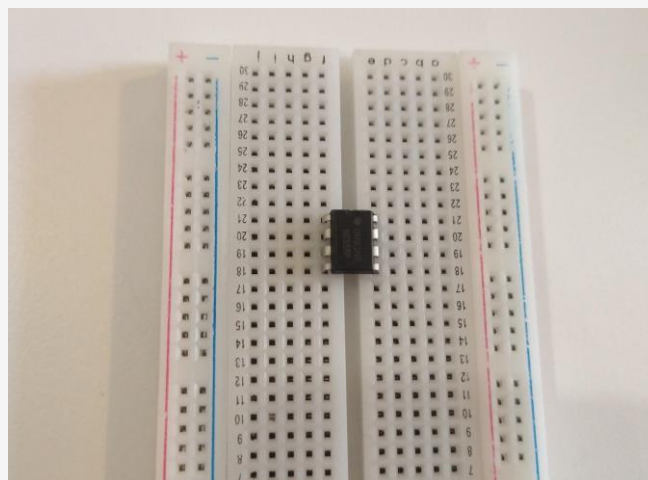
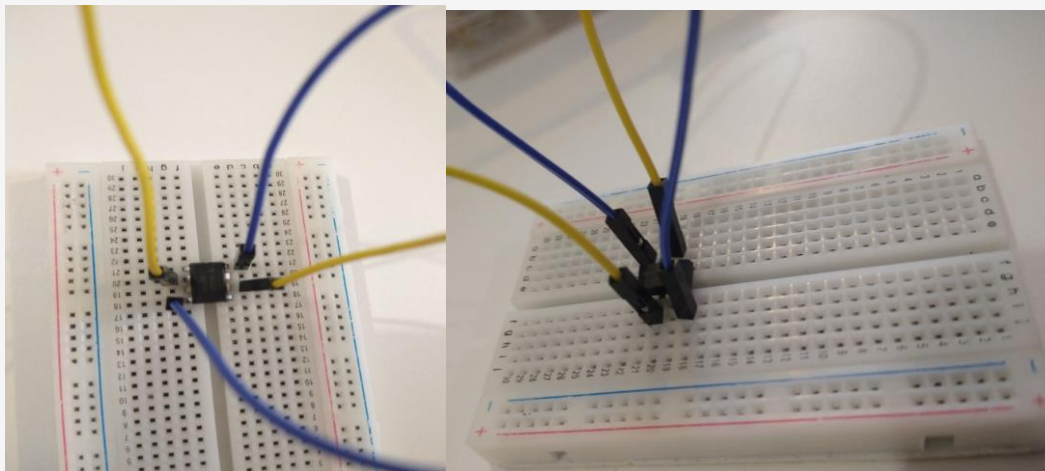


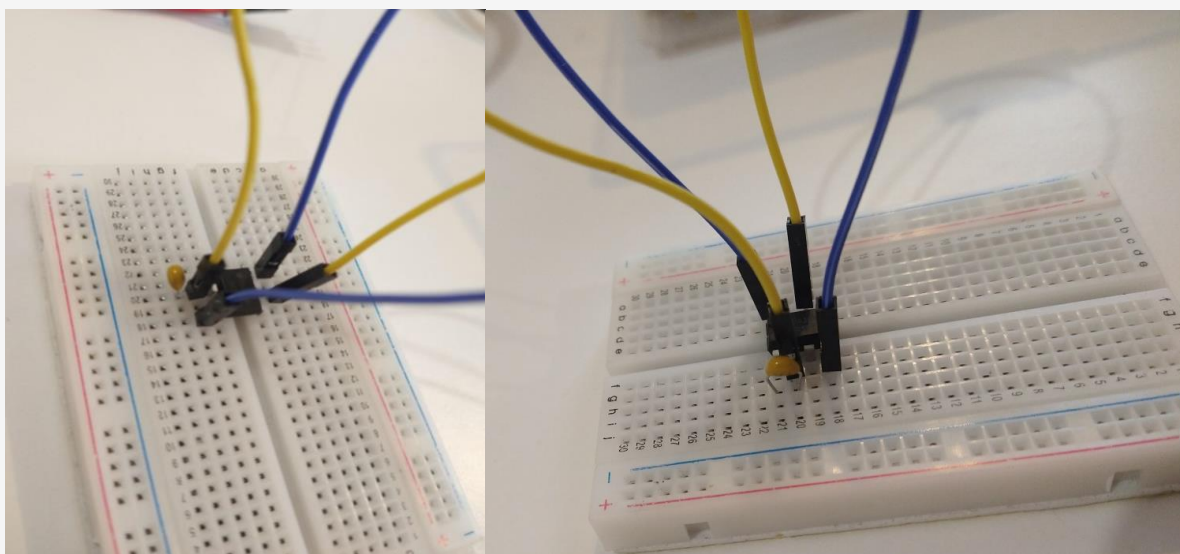
Figure 46 – Timer-pinout

Source: <http://www.learningaboutelectronics.com/Articles/555-timer-pinout.php>

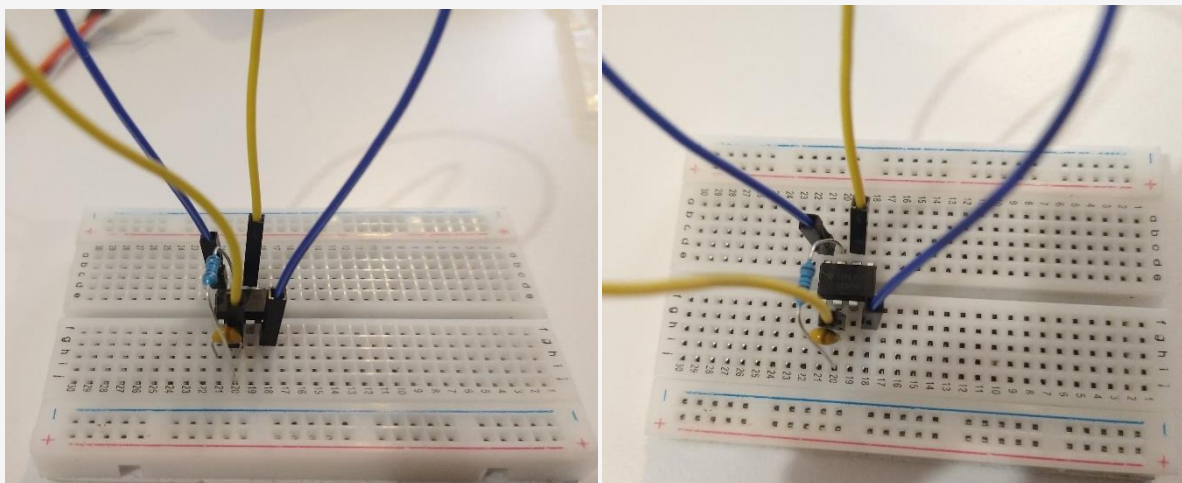
Then we add two jumper wires, one that connects leg 4 and leg 8 of the NE555 and another that connects legs 2 and 6.



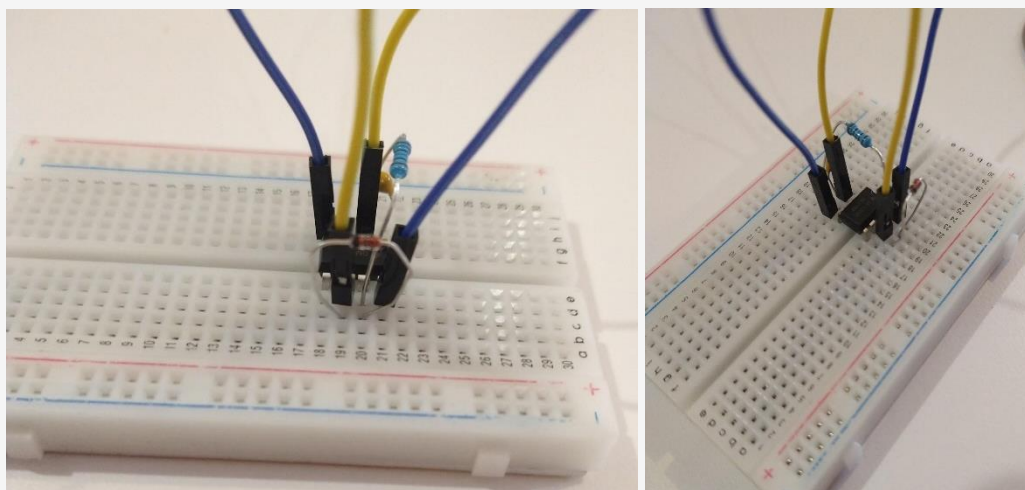
Next, we add a capacitor that connects legs 1 and 2.



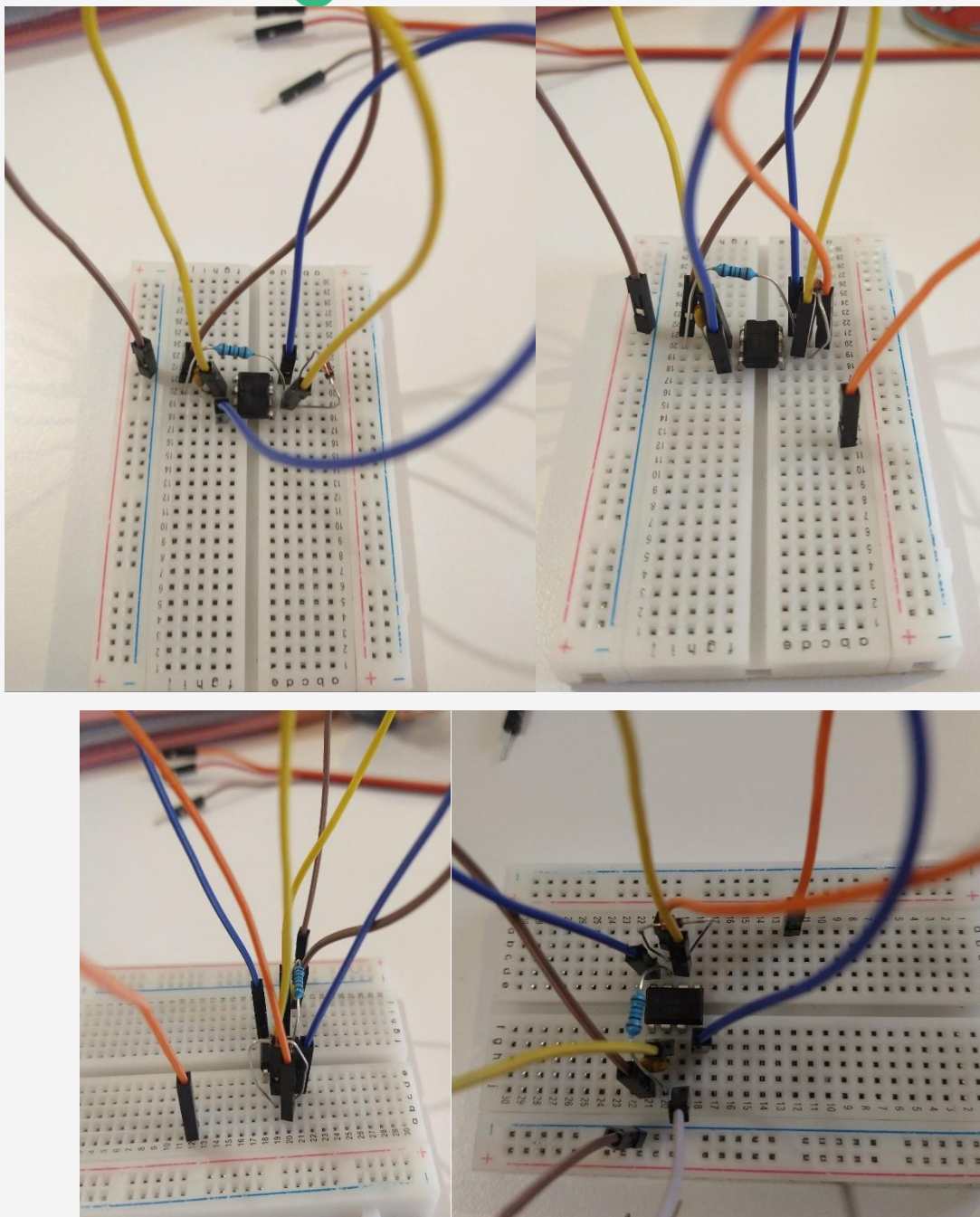
We add a 220k ohm resistor that connects legs 2 and 7.



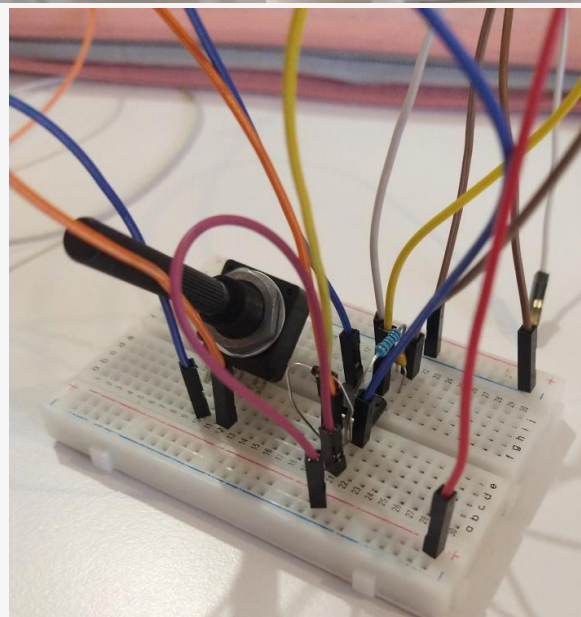
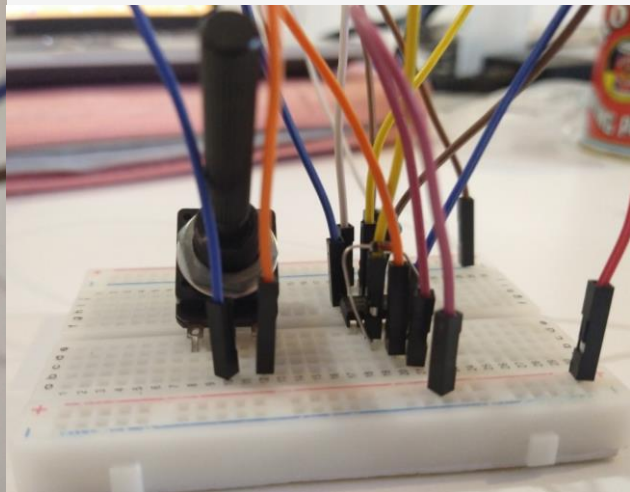
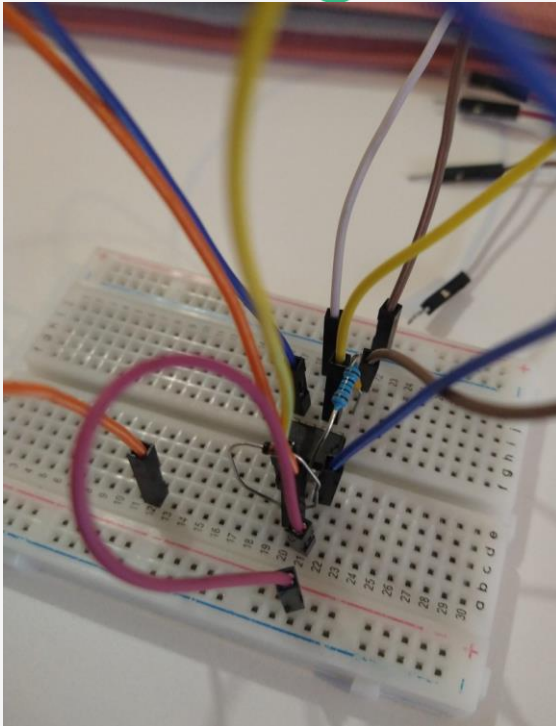
Then we add a Zener diode that connects legs 6 and 7 of the NE555 as shown in the photos. Make sure to put the diode in the right direction, i.e., with the black stripe on leg 6.



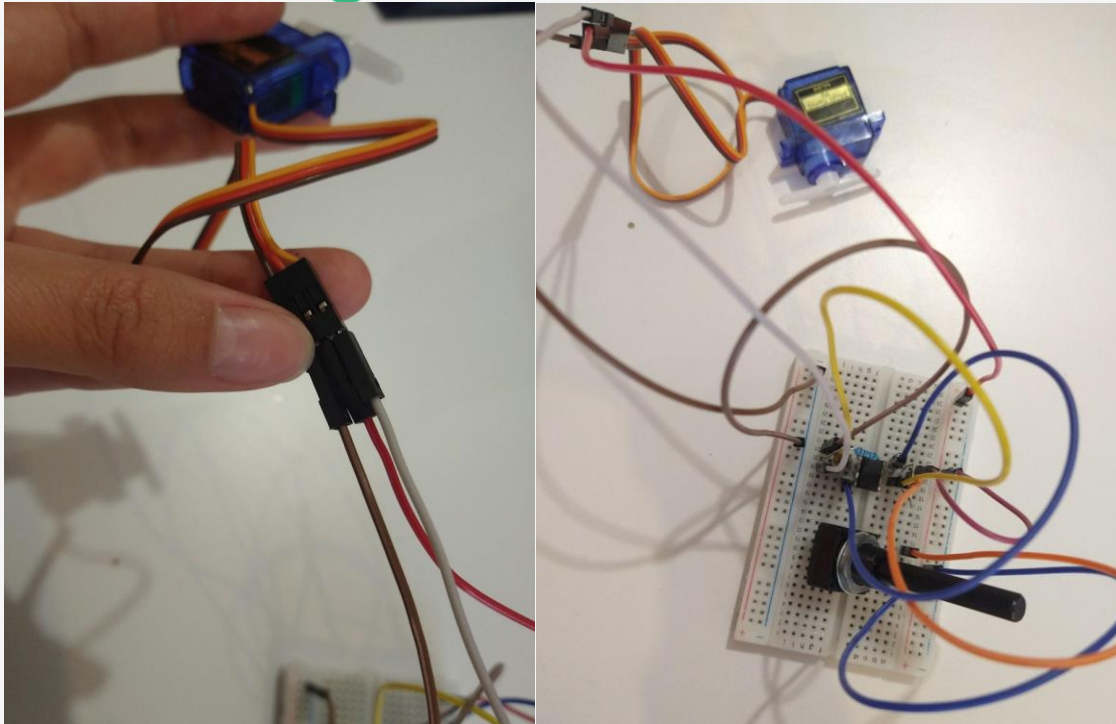
We add a jumper wire that goes from leg 1 to the negative pole and then another that connects leg 7 to another line further in the breadboard. A new wire will be connected on one side to leg 3 of the NE555 (for the moment we do not take care of the other end of the wire).



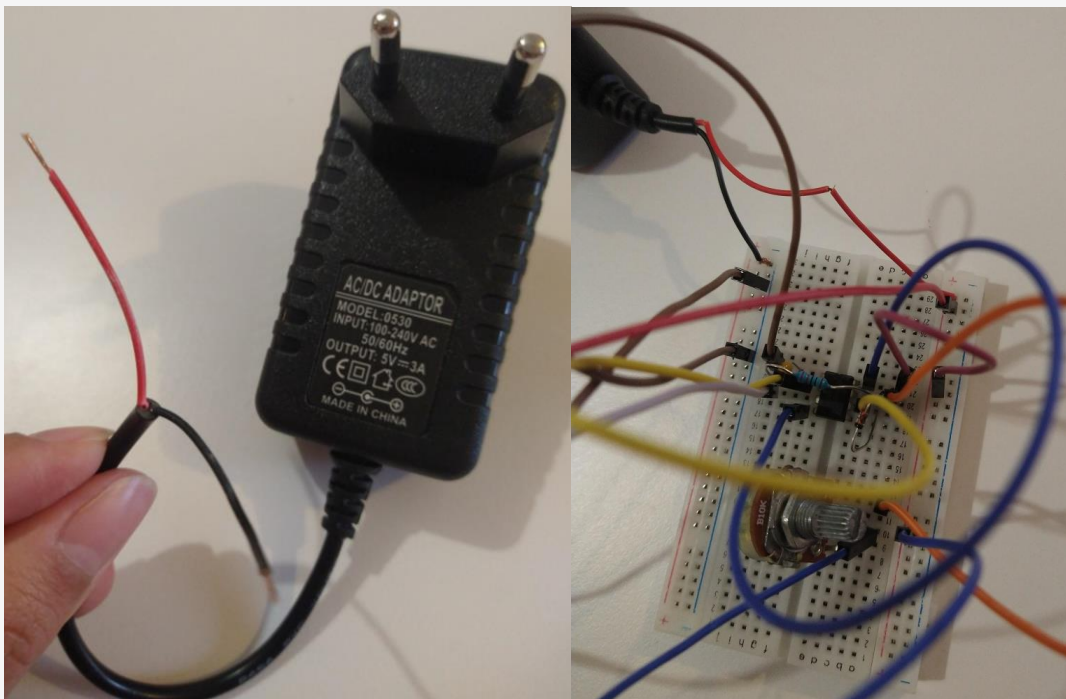
We add a jumper wire that connects leg 1 to the positive pole. Then, we position the potentiometer as in the photo below. With a leg on the same line of the cable (orange) that started from leg 7 of the NE555. We add on the line where the middle leg of the potentiometer is located, another wire which will join the positive pole.



We now take the (white) jumper wire that was connected to leg 3 of the NE555 from one end and we connect it to the orange cable of the servo motor. Also, we connect a wire to the positive pole of the breadboard from one end to the red wire of the servo on the other end. We do the same with a wire that comes out of the negative pole of the breadboard and connects to the brown cable of the servo motor (See pictures below).

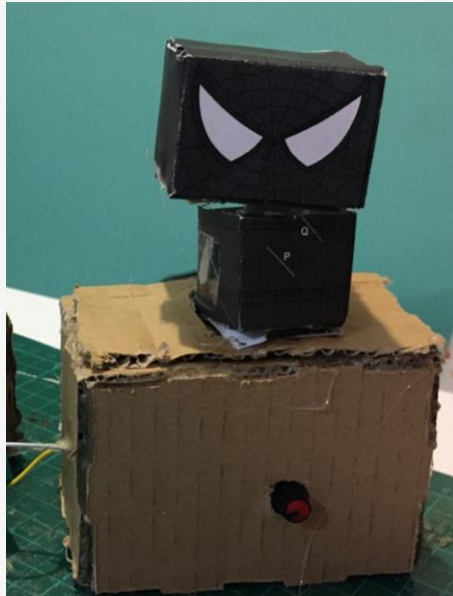


The last step is to take a 5V adapter, cut the tip and remove the plastic to have the red cable and the black cable available. We strip a little the two ends that we will also introduce into the positive and negative poles respectively.



That's it, our circuit is finished!

To complete the project, connect the servo motor to the head of the paper toy and hide the electronic circuit inside a cardboard box. Make sure to drill a hole for the potentiometer.



3.6. Remote-controlled door lock

IR, or infrared communication is a common, inexpensive and easy to use wireless communication technology. IR light is very similar to visible light, except that it has a slightly longer wavelength. This means IR is undetectable to the human eye - perfect for wireless communication.

In order to understand how IR technology works, in the following section you will be building a door lock connected to an Arduino board.

Build a remote-controlled door lock that functions with infrared technology

In this project we will be using the information gathered previously about infrared technology to create a remote-controlled door lock.

You will be using an infrared remote controller, an infrared receiver and a servomotor to complete this project.

Step 1: Install the library

Head to [this](#) website and download the library. Next, we need to install the library on the Arduino IDE software. Go to Arduino IDE → Sketches → Include a library → IRremote.

Step 2: Figure out the key of your remote

In order to figure out how your arduino board interprets the electric signals sent out by your remote, we need to upload the following code onto the board.

```

/* Finding the key codes for your remote. More info: https://www.makerguides.com */

#include <IRremote.h> // include the IRremote library

#define RECEIVER_PIN 13 // define the IR receiver pin
IRrecv receiver(RECEIVER_PIN); // create a receiver object of the IRrecv class
decode_results results; // create a results object of the decode_results class

void setup() {
  Serial.begin(9600); // begin serial communication with a baud rate of 9600
  receiver.enableIRIn(); // enable the receiver
  receiver.blink13(true); // enable blinking of the built-in LED when an IR signal is received
}

void loop() {
  if (receiver.decode(&results)) { // decode the received signal and store it in results
    Serial.println(results.value, HEX); // print the values in the Serial Monitor
    receiver.resume(); // reset the receiver for the next code
  }
}

```

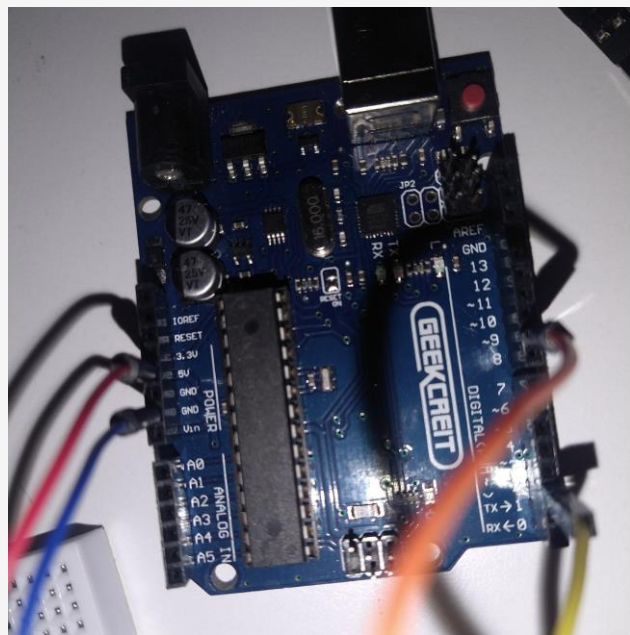
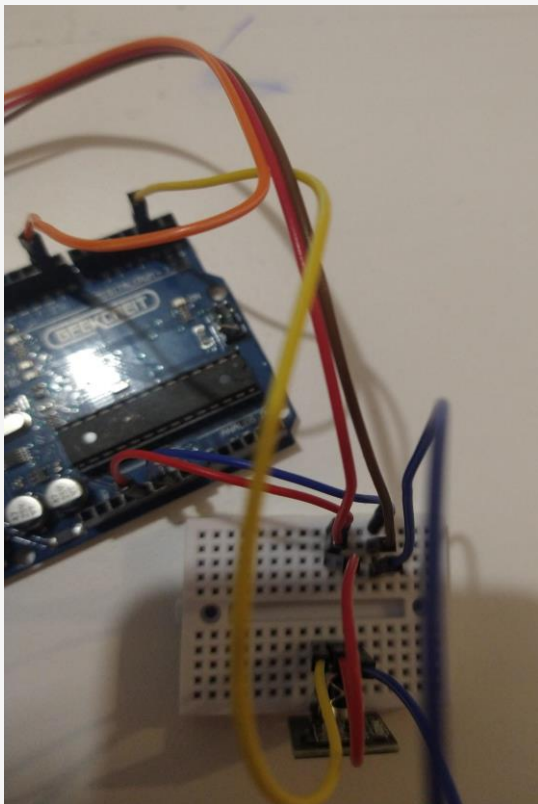
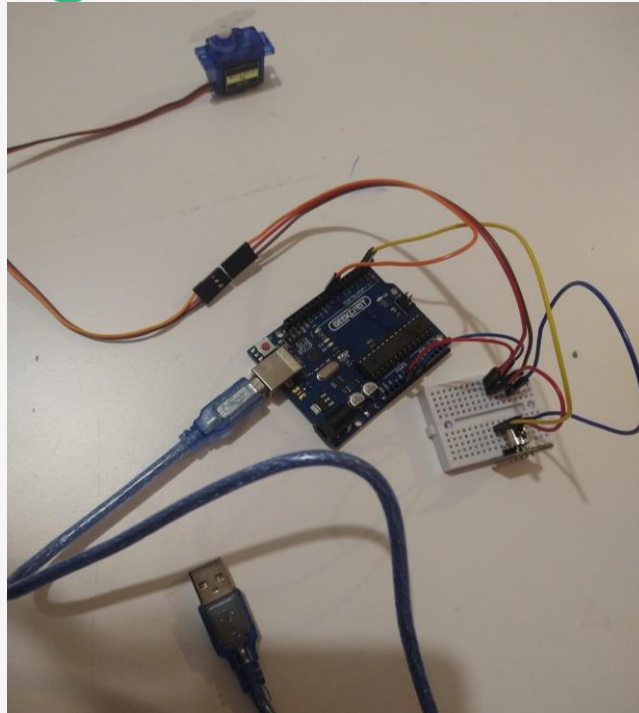
Next, you can open the Arduino serial monitor and by clicking on the remote buttons you will be able to view the key that gets displayed on the serial monitor. Each button of your remote corresponds to a different key. Take note of the different keys because you will need this information later.

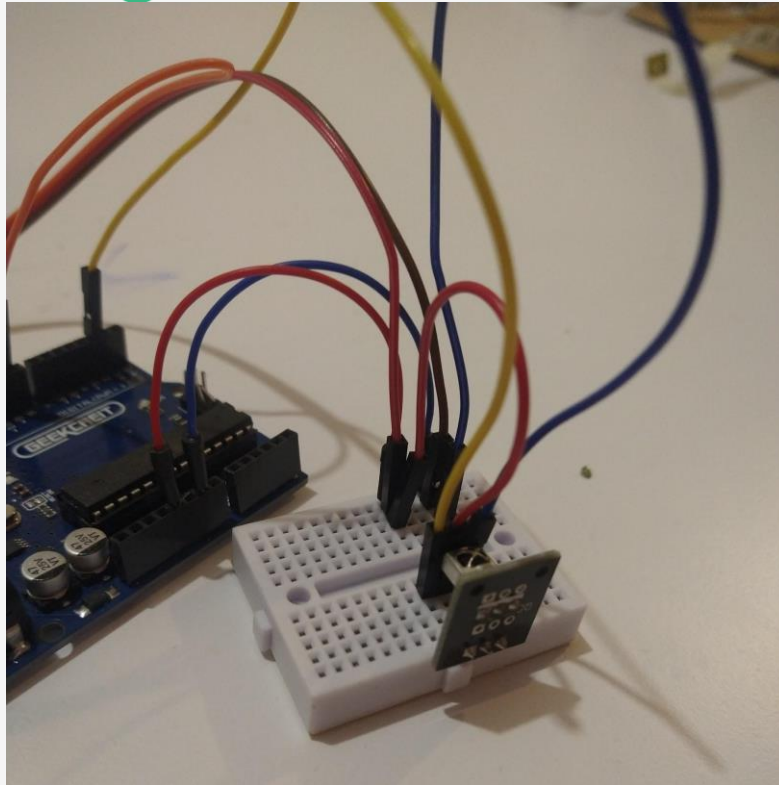
Step 3: Wire all components

Wire all components as illustrated in the images below. Be careful with the IR receive: the positive output goes to 5V of the Arduino board, the negative output to ground and the signal output to digital pin 2 (see code below).

Notice that the signal output of the servomotor is connected to digital pin 9 of the Arduino board.







Step 4: Code

We need to upload the following code onto the board:

```
#include <IRremote.h> //must copy IRremote library to arduino libraries
#include <Servo.h>
#define up 0xFF906F //clockwise rotation button
#define down 0xFFE01F //counterclockwise rotation button

int RECV_PIN = 2; //IR receiver pin
Servo servo;
int val; //rotation angle
bool cwRotation, ccwRotation; //the states of rotation

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
  servo.attach(9); //servo pin
```

```

}

void loop()
{
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume(); // Receive the next value

    if (results.value == up)
    {
      cwRotation = !cwRotation; //toggle the rotation value
      ccwRotation = false; //no rotation in this direction
    }

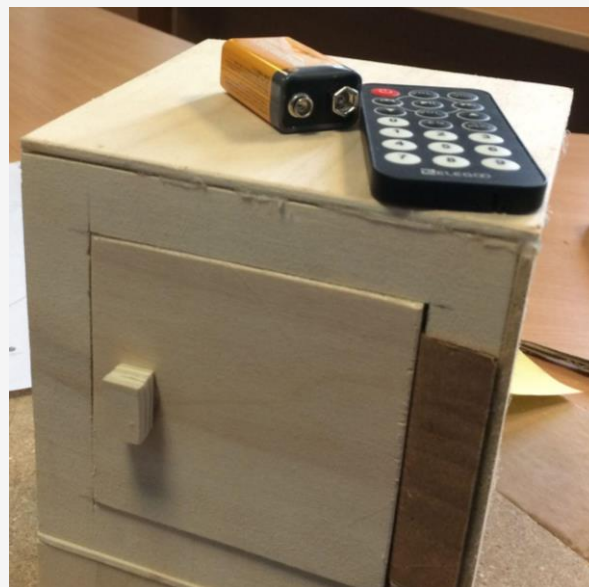
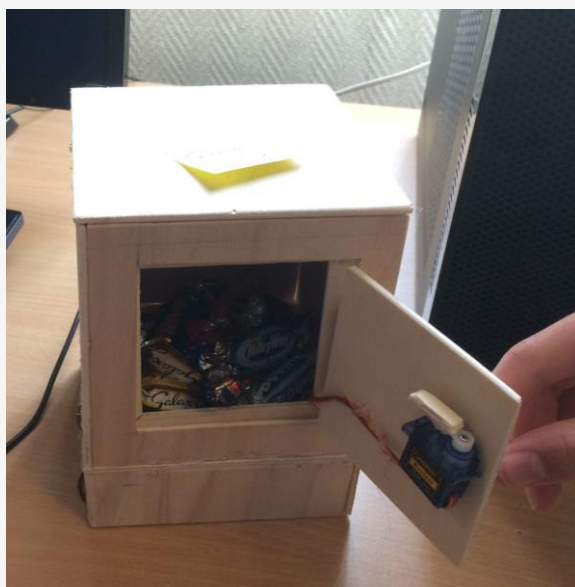
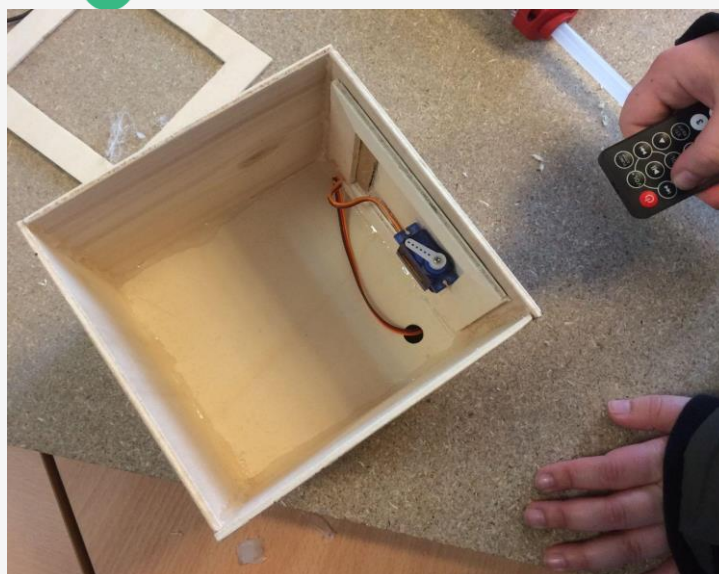
    if (results.value == down)
    {
      ccwRotation = !ccwRotation; //toggle the rotation value
      cwRotation = false; //no rotation in this direction
    }
  }
  if (cwRotation && (val != 175)) {
    val++; //for clockwise button
  }
  if (ccwRotation && (val != 0)) {
    val--; //for counter clockwise button
  }
  servo.write(val);
  delay(20); //General speed
}

```

Step 5: Create the door lock

By now you should have tested your project. If the servo motor responds to the commands sent by the remote controller, then it's time to think about the door lock. You can be creative here and imagine any type of door lock. We suggest a solution that involves attaching the servo motor directly to a door, like in the image below.





3.7. Measuring temperature, humidity, light and colour

In the previous projects, we have used motors to achieve different results, from simply driving a motor clockwise or counterclockwise to building more sophisticated gadgets such as the interactive paper toy or the remote-controlled door lock.

Now, motors are actuators which means that they are meant to perform a certain action. Together with actuators, one could also choose to employ some sensors. We've actually explored a couple of sensors in the previous chapter. The first one was the potentiometer, the second one was the infrared sensor which receives IRR signals and translates them into computer code for the Arduino board.

This section is all about sensors. The objective is to familiarize yourself with this type of electronic components to be able to take your own projects to the next level.

In particular, we will be exploring sensors that measure temperature, light, as well as humidity and colour.

3.7.1. Using a sensor with Arduino

The light sensor is a type of resistor, it is called a light-dependent resistor. It is used to detect light and also to measure the brightness level of a certain environment.

A light sensor has two pins and because it is basically a resistor we do not need to distinguish between these two pins.

The higher the intensity of the light, the smaller is the resistance recorded by the light sensor. Therefore, by measuring the light sensor resistance we can know how bright the environment is.

This is how to wire a photoresistor to an Arduino board.

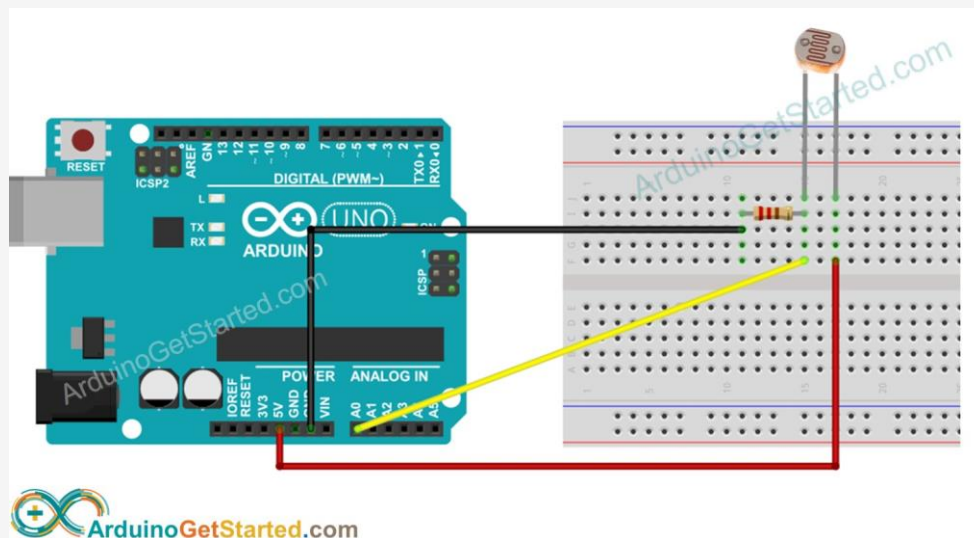


Figure 47 – Connecting a photoresistor to an Arduino board

Source: <https://arduinogetstarted.com/tutorials/arduino-light-sensor>

And this is a simple code that allows you to view the values recorded by the light sensor via the serial monitor of the Arduino IDE.

```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
void loop() {
  // reads the input on analog pin A0 (value between 0 and 1023)
```

```

int analogValue = analogRead(A0);

Serial.print("Analog reading: ");

Serial.print(analogValue); // the raw analog reading

delay(500);

}

```

3.7.2. Build a theremin with Arduino and a light sensor

The theremin is a synthesizer that makes a sound as you wave your hands in front of it. It is basically an electronic musical instrument.

In this project, we are going to make a similar instrument that will change the pitch of the note as you wave your hand in front of it.

We will need a piezo buzzer, a light sensor and a 1k ohm resistor.

Step 1: Wiring

Complete the wiring as per the diagram below

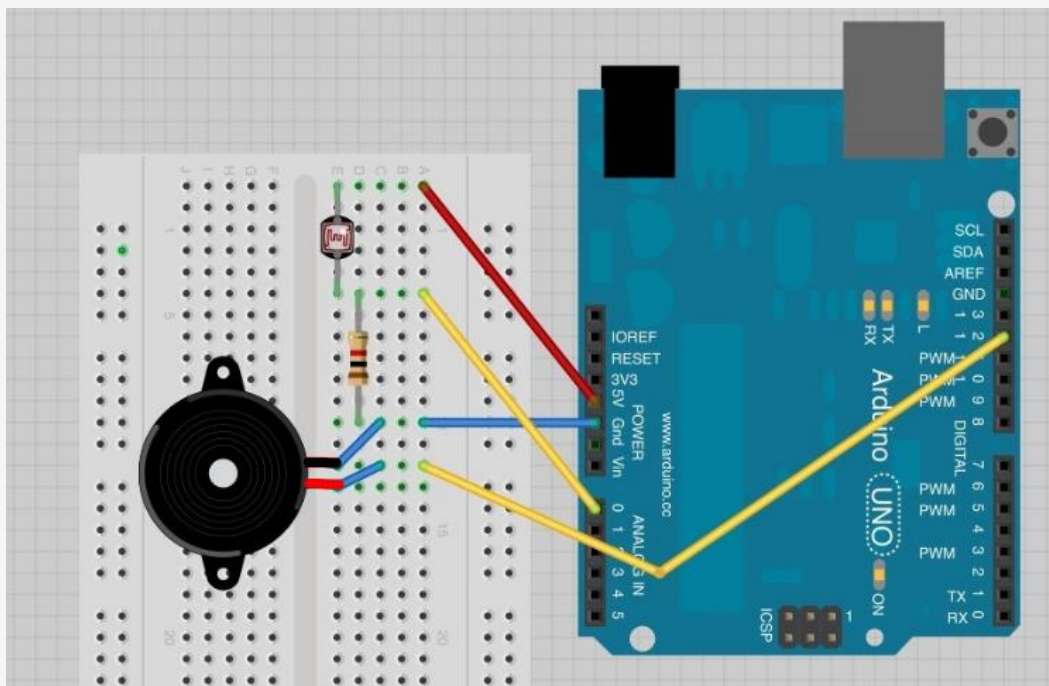


Figure 48 – Wiring a theremin with Arduino

Source: <https://learn.adafruit.com/adafruit-arduino-lesson-10-making-sounds/pseudo-theramin>

Step 2: Code

```
int speakerPin = 12;

int photocellPin = 0;

void setup()
{
}

void loop()
{
  int reading = analogRead(photocellPin);

  int pitch = 200 + reading / 4;

  tone(speakerPin, pitch);
}
```

The sketch is actually really straightforward. We simply take an analog reading from A0, to measure the light intensity. This value will be in the range of something like 0 to 700.

We add 200 to this raw value to make 200 Hz the lowest frequency and simply add the reading divided by 4 to this value to give us a range of around 200 Hz to 370 Hz.

In order to produce different effects, you can try to change the value by which the reading of the light sensor is divided. For example, replace 4 with 2 and see what happens.

3.7.3. Colour sensitive robots

As the name suggests, colour sorting consists in sorting objects according to their colour.

Obviously, this can be achieved by looking at each object and deciding to put it at one place or another, however, when objects become too many this can be a tedious and incredibly repetitive task. Then automatic colour sorting machines can come in very handy.

These machines have colour sensors to sense the colour of any object. After detecting the colour, a motor or system of motors grabs the object and places it into the respective receptacle. Colour sorting machines can be used in different areas where colour identification, colour distinction and colour sorting are important. Some of the application areas include the Agriculture Industry (Grain Sorting on the basis of colour), the Food Industry, the Diamond and Mining Industry, Recycling etc.



Industrial colour sorter machines

Let's take a look at some industrial colour sorter machines.



Figure 49 - Industrial colour sorter machines.

Source: <http://hugeacademy.com>

Sorters can be divided into chute-type and belt-type colour sorters.

Belt-type colour sorters break a smaller percentage of the material (important for nuts) and the product stays relatively static during the transport process as it moves horizontally on the belt. In the chute type, material slides on the chute because of gravity causing collision, friction and larger vertical movements, thus worsening the ratio of broken material. The belt structure makes the transmission smooth and stable without bouncing of material.

Chute-type colour sorters are more common especially for food as prices are lower, capacities are higher and products can be seen more easily from both sides, which is important when a dehulled grain has a hull only on one side. Chute sorters are usually applicable to specific products, as the chute is designed with special channels for this kind of material based on sizes and shapes of the material. For example, 5 mm chutes are used for rice, grain and plastic granules. Flat chutes are right for plastic flakes, such as PET or milk bottle flakes.

3.7.4. Introduction to DHT11 sensor

There is a sensor for just about everything, including to measure temperature!

The DHT11 sensor is a temperature and humidity sensor, which means that it can measure both parameters. In this project, we learn how to wire a DHT11 sensor to an Arduino board and how to program the board in order to view the temperature values that are being recorded by the sensor in real-time.

Step 1: Wiring

Wire all components as per the image below:

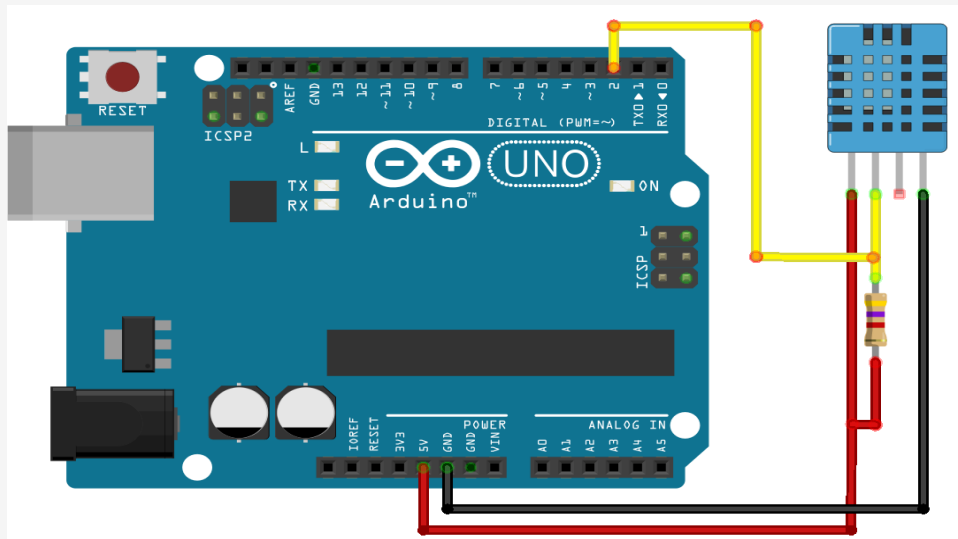


Figure 50 - DHT11 sensor on Arduino UNO

Source: <https://randomnerdtutorials.com/complete-guide-for-dht11dht22-humidity-and-temperature-sensor-with-arduino/>

It is possible to ignore the resistor in the diagram, which is a 4.7 kohm resistor.

Step 2: Installing libraries

To read from the DHT sensor, we'll use the DHT library from Adafruit. To use this library you also need to install the Adafruit Unified Sensor library. Follow the next steps to install those libraries.

Open your Arduino IDE and go to Sketch > Include Library > Manage Libraries. The Library Manager should open.

Search for "DHT" on the Search box and install the DHT library from Adafruit.

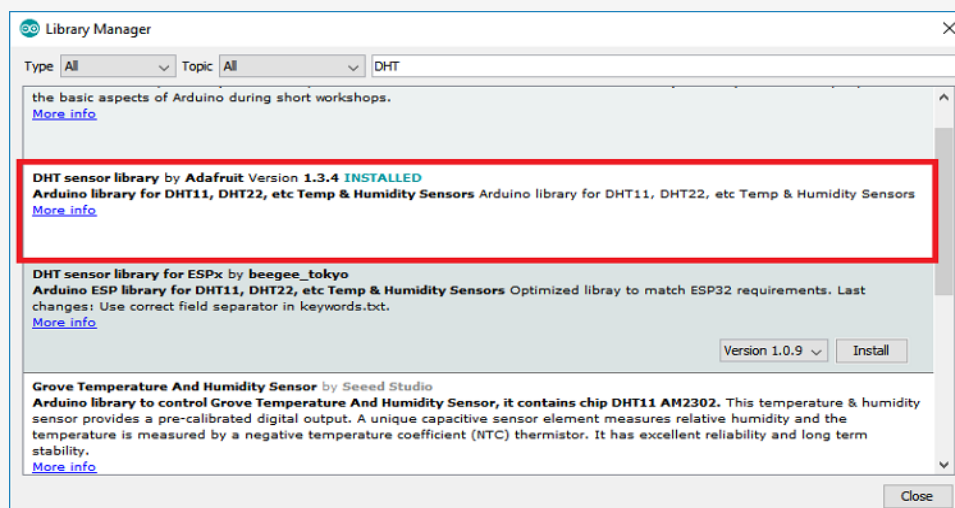


Figure 51 – Installing DHT library from Adafruit

Source: <https://randomnerdtutorials.com/complete-guide-for-dht11dht22-humidity-and-temperature-sensor-with-arduino/>

Step 3: Code

```
#include "DHT.h"

#define DHTPIN 2 // what pin we're connected to

#define DHTTYPE DHT11 // DHT 11

// Initialize DHT sensor for normal 16mhz Arduino

DHT dht(DHTPIN, DHTTYPE);

void setup() {

  Serial.begin(9600);

  Serial.println("DHTxx test!");

  dht.begin();
}

void loop() {

  // Wait a few seconds between measurements.

  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)

  // Read temperature as Celsius

  float t = dht.readTemperature();

  Serial.print("Temperature: ");

  Serial.print(t);

  Serial.print(" *C ");

}
```



3.7.5. Build a smart cooling fan

In this project, you will be coupling the knowledge on sensors and actuators to create a useful and unique gadget: an intelligent cooling fan. You will be using a DC motor driven by a motor shield (we've explored this in a previous chapter) plus a DHT11 sensor to record the temperature of your room.

The final result is a cooling fan that operates when a certain temperature threshold is attained, of course you will be able to program the system as you please and determine this threshold temperature yourself.

Step 1: 3D printing the cooling fan

We've found [this](#) design on thingiverse. It is a mini cooling fan that uses a DC motor. It is also possible to 3D print a battery case to house a 9v battery. However, this won't be necessary for our purposes as we will be using the 9v battery to power the motor shield.

Have a go at printing the fan and the support. If you don't have access to a 3D printer it is of course possible to create these two items out of cardboard or plywood or any other material that's consistent enough.

Step 2: Wiring it all up

Wire up all the components as per the image below:

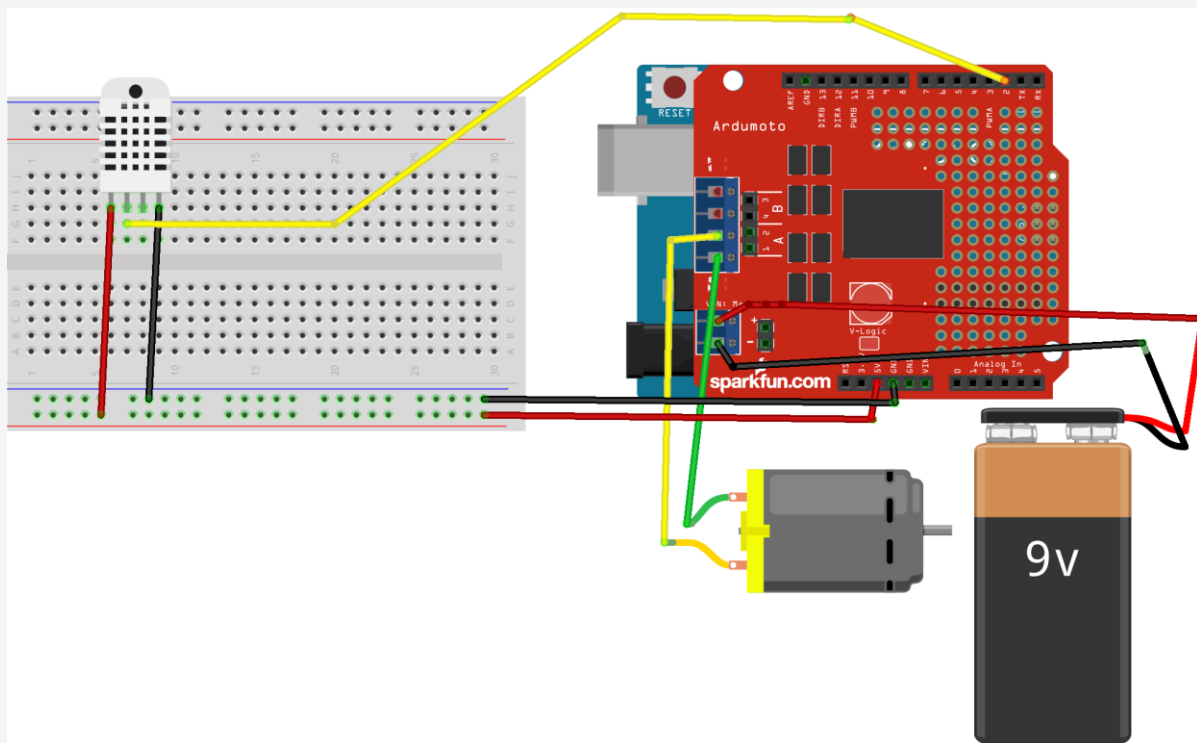


Figure 52 – Wiring the smart cooling fan

Source: [Digijeunes](#)

Step 3: Code

With this program, we are instructing the Arduino board to run the DC motor if the temperature recorded by the DHT11 sensor is equal to or greater than 24°. Otherwise, the DC motor will not spin.

```
#include "DHT.h"

#include "AFMotor.h"

#define DHTPIN 2 // what pin we're connected to

AF_DCMotor motor1(1); // We define a motor attached to M1 on the motorshield

#define DHTTYPE DHT11 // DHT 11

// Initialize DHT sensor for normal 16mhz Arduino
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);

  motor1.setSpeed(100); // We define the speed with which the motor will spin

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (it's a very slow sensor)
```



```
// Read temperature as Celsius
float t = dht.readTemperature();

Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" *C ");

if (t >= 24)
{
  motor1.run(BACKWARD);
}
else
{
  motor1.run(RELEASE);
}
}
```

References

- Autodesk, Inc. (2020). *What You'll Learn*. <https://www.instructables.com/Tools-andMaterials-for-Arduino/>
- Boxall, J. (2013). *Arduino workshop: A Hands-On introduction with 65 projects*. No Starch Press.
- Circuit Basics (n.d.). *Introduction to Microcontrollers*. <https://www.circuitbasics.com/introduction-to-microcontrolleres/>
- Green Steam Incubator. (2019). Module on Microcontrollers: 30 hours lessons. <https://steam-incubator.org/wp-content/uploads/2021/11/IO3.2-GSI-Module-on-Microcontrollers.pdf>
- Makerspaces.com (2022). *Arduino For Beginners*. <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>
- Techatronic (2022). *Types of Arduino Boards*. <https://techatronic.com/types-of-arduino-boards-arduino-uno-mega-mini-specification/>



- Learn Adafruit (2022). *Adafruit motor shield*. <https://learn.adafruit.com/adafruit-motor-shield/af-dcmotor-class>
- Sparkfun (2022). *Servos*. <https://www.sparkfun.com/servos>
- Learning about electronics (2022). *555 timer pinout*.
<http://www.learningaboutelectronics.com/Articles/555-timer-pinout.php>
- Girls in STEM (2022). EU funded project. <https://girlsinstem.eu/>
- Arduino Get Started (2022). *Arduino light sensor*.
<https://arduinogetstarted.com/tutorials/arduino-light-sensor>
- Adafruit (2012). *Pseudo theremin*. <https://learn.adafruit.com/adafruit-arduino-lesson-10-making-sounds/pseudo-theramin>
- Huge academy (2022). <http://hugeacademy.com/>
- Random nerd tutorials (2022). *Complete guide for DHT11 humidity and temperature sensor with Arduino*. <https://randomnerdtutorials.com/complete-guide-for-dht11dht22-humidity-and-temperature-sensor-with-arduino/>
- DIY robotics (2020). *Articulated robots*. <https://diy-robotics.com/article/articulated-robots/>
- DIY robotics (2020). *What you should know about cartesian robot*. <https://diy-robotics.com/article/what-you-should-know-about-cartesian-robot/>
- DIY robotics (2020). *Scara robots*. <https://diy-robotics.com/article/scara-robots/>
- DIY robotics (2020). *Articulated robots*. <https://diy-robotics.com/article/top-six-types-industrial-robots-2020/>
- Learn mech (2022). *Cylindrical robot diagram construction applications*.
<https://learnmech.com/cylindrical-robot-diagram-construction-applications/>
- How to robot (2022). *Industrial robot types and their different uses*.
<https://www.howtorobot.com/expert-insight/industrial-robot-types-and-their-different-uses>
- Robot Worx (2022). *What are the main types of robots*. <https://www.robots.com/faq/what-are-the-main-types-of-robots>
- Built In (2022). *Robotics*. <https://builtin.com/robotics>
- Analytics Insight (2021). *Common types of robots*.
<https://www.analyticsinsight.net/common-types-of-robots-are-there-any-new-ones-you-havent-heard-yet/>
- Stanford Edu (2022). *Army robots*
<https://cs.stanford.edu/people/eroberts/cs201/projects/2010-11/ComputersMakingDecisions/army-robots/index.html>
- NCBI (2019). *Articles*. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6625162/>
- Guardforce (2022). <https://www.guardforce.com.hk/>
- Iberdrola (2022). *Educational robots*. <https://www.iberdrola.com/innovation/educational-robots>





Co-funded by
the European Union

The #CodER project is co-financed by the ERASMUS+ programme of the European Union and is implemented from December 2021 to November 2023. This publication reflects the views of the authors and the European Commission cannot be held responsible for any use which may be made of the information contained therein

Project Number: 2021-1-FR02-KA220-YOU-000028696

