Co-funded by the
Erasmus+ Programme
of the European Union

Module 2

General Introduction to Computational Thinking

A basic module suitable for all teachers

Authors: Radboud University (Netherlands)
Maria Kallia,
Sjaak Smetsers,
Erik Barendsen,
Christos Chytas

Reviewers:
Arnold Pears (KTH),
Valentina Dagienė (VU)

External Reviewers:
Piret Luik (Estonia),
Renate Motschnig (Austria)

Piloting:
Ankara University (Turkey), KTH Royal Institute of Technology Sweden), Radboud University (Netherlands), University of Paderborn (Germany), Vienna University of Technology (Austria)

Design (icons):
Vaidotas Kinčius (Lithuania)

Module *2* is based on the work within the project "Future Teachers Education: Computational Thinking and STEAM" (TeaEdu4CT). Coordination: Prof. Valentina Dagienė, Vilnius University, Lithuania. Partners: Vienna University of Technology (Austria), CARDET (Cyprus), Tallinn University (Estonia), University of Turku (Finland), Paderborn University (Germany), CESIE (Italy), Radboud University (Netherlands), KTH Royal Institute of Technology (Sweden), Ankara University (Turkey). The project has received co-funding by the Erasmus+ Programme KA2.

Content

General overview and aim 3

Target groups and prerequisites 4

Learning Outcomes and Assessment Methods 4

Module plan and didactical approaches 4

Units and activities 5

UNIT 1: Introduction to Computational Thinking 5

UNIT 2: Computational Thinking Tools 8

UNIT 3: Experience Computational Thinking via programming 25

UNIT 4: Teaching and Learning Computational Thinking 39

Overview of Learning Resources of Module O2 46

General overview and aim

This module aims to provide prospective teachers with a concrete understanding of computational thinking (CT), working knowledge of teaching and learning principles for CT, and to introduce them to computational tools that can support teaching and learning.

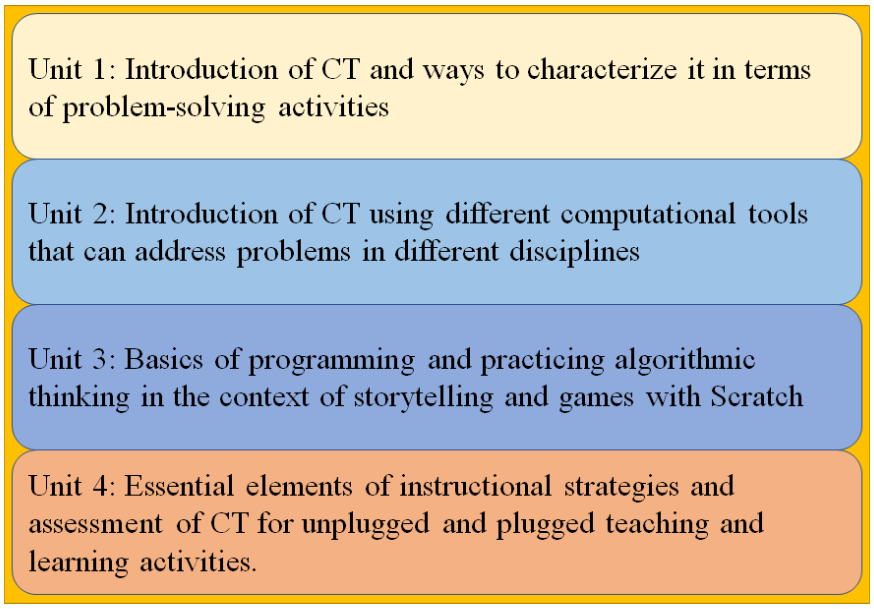
In this module, the learners will:

- explore the concept of computational thinking and compare various ways to characterize it in terms of problem-solving activities
- engage with computational thinking aspects, practices and tools and understand the role of computational thinking within the disciplines
- get acquainted with the basics of programming and will practice algorithmic thinking in the context of storytelling and games
- experience both unplugged and plugged teaching and learning activities and learn about design principles for instructional strategies for computational thinking

The module considers CT as a framework to develop cross-curricular skills and competences, suitable for any subject teacher.

The module structure

To this end, the module is organised into the following four units: Unit 1 introduces the concept of CT and compares ways to characterise it in terms of problem-solving activities; Unit 2 introduces CT by utilising different computational tools (Ngrams, NetLogo, Excel) and demonstrates how these tools can be employed to address problems in different disciplines (History, Biology, Geography); Unit 3 focuses on the basics of programming and on practicing algorithmic thinking in the context of storytelling and games with Scratch. The module concludes with Unit 4, which highlights unplugged and plugged teaching and learning activities and discusses essential elements of instructional strategies and assessment for CT.



Unit 1: Introduction of CT and ways to characterize it in terms of problem-solving activities

Unit 2: Introduction of CT using different computational tools that can address problems in different disciplines

Unit 3: Basics of programming and practicing algorithmic thinking in the context of storytelling and games with Scratch

Unit 4: Essential elements of instructional strategies and assessment of CT for unplugged and plugged teaching and learning activities.

Target groups and prerequisites

This module is for prospective teachers studying in a teacher education programme as well as for the professional development of in-service teachers interested in computational thinking. The module is designed for face-to-face learning, but it can easily be adapted as a distant learning module.

There are no particular prerequisites for studying this module. It would be advisable for students to have a basic knowledge of the tools employed and having completed the previous module "O1: Frameworks for the support of the modules: CT&STEM for future teacher education".

Learning Outcomes and Assessment Methods

A successful learner who have completed the whole module, will be able to::

- explain the concept of computational thinking and compare various ways to characterize it in terms of problem-solving activities;
- carry out problem solving tasks using computational thinking concepts, practices and tools and explain the role of computational thinking within the disciplines;
- construct simple programs in Scratch and apply algorithmic thinking in the context of storytelling and games;
- apply unplugged and plugged teaching strategies and learning activities for computational thinking and describe the underlying design principles.

More specific learning goals can be found within the structure of the units.

Assessment Strategy

The tasks throughout the modules are opportunities for formative assessment and feedback. There is an optional final (overall) assessment which can be found as a separate document (see learning resources: Final assessment module).

Module plan and didactical approaches

The module consists of 4 units of face-to-face interaction. Each unit comprises several activities which usually start with a warming-up activity and conclude with a reflective activity. The students will engage with a variety of learning approaches which include reading articles, or the provided literature reviews, group discussions, problem solving tasks, and reflection activities.

Units and activities

Unit 1: Introduction to Computational Thinking

Activity 1.1 Introduction to CT

- Computational thinking as problem solving activity
- Characterizing Computational Thinking elements
- Conclusion

Total: 2 hours

Unit 2: Computational Thinking Tools

Activity 2.1: Using Google Ngrams

Activity 2.2: Modelling and Simulation through NetLogo

Activity 2.3 Using Microsoft Excel

Total hours: 8 hours

Unit 3: Experience Computational Thinking via Programming

Activity 3.1: Storytelling in Scratch

Activity 3.2: Creating a maze game

Total hours: 10 hours

Unit 4: Teaching and Learning Strategies for Computational Thinking

Activity 4.1: Bebras Tasks

UNIT 1: Introduction to Computational Thinking

In this unit you will explore the concept of computational thinking and compare various ways to characterize it in terms of problem-solving activities.

Contribution to the learning outcomes

Learning outcomes
● be able to describe computational thinking as a connecting mechanism between a subject area and IT

● characterize computational thinking in terms of problem-solving steps and activities leading to an operational, executable solution
● recognize elements of computational thinking in scenarios for students' activities

Activity 1.1 Introduction to Computational Thinking

To put the full potential of computers to use in a variety of subjects and activities, more digital skills are required than being able to operate a program like Word or maintain a social media account. Establishing such a connection between subject matter and information technology requires specific problem-solving skills. Those skills are in the domain of Computational Thinking

Computational thinking as a problem solving activity

Reading Task

Read the document 'A brief introduction to computational thinking' which can be found in the learning resources. This text distinguishes three main steps in computational thinking: the arrows (1), (2) and (3) in the diagram.

Pair Discussion Task

Discuss the two classroom scenarios below, taken from Yadav et al. (2018, p. 381). Which activities would you consider as computational thinking? How do they relate to the steps (1), (2) and (3) in Materials A?

Scenario 1:

Westwood Elementary school will start the next school year with a 1:1 iPad initiative. Mr. Nowak has decided to have his 2nd grade students use their iPads to predict weather (temperature, precipitation, and wind) for a week. Each student draws a picture of what they think the weather will look like. Sara, a student, also wanted to keep track of the temperatures that everyone predicted. Mr. Nowak started a Google spreadsheet where each student entered their predicted temperatures. The next day, they recorded the actual weather by using Accuweather App on their iPads and entering the information in the Google sheet. Olivia also wanted to record the actual temperature in Sara's spreadsheet so that they could compare how their predictions compared to what the weather actually was. After a week, they projected the Google spreadsheet on the smartboard and subtracted the differences between the observed and predicted temperatures. Mr. Nowak demonstrated how to make a bar graph of those differences.

Scenario 2:

All the second-grade classes are taking a field trip! The school cafeteria packed PB&J lunches for everyone in identical paper bags, except for Sara and Olivia who have are allergic to peanuts. The lunch paper bags are labelled with all the student names and divided them up into 10 boxes with 10 lunches per box. The lunches were placed in boxes in alphabetical order by last name. Mr. Nowak wants to check to be sure that Sara and Olivia receive peanut-free lunches. They help him search through the boxes. Olivia Velazquez knows that her lunch will probably be near the end, so she looks at the first lunch in each box until she finds one starting with a letter close to the end of the alphabet. When she finds the box that begins with Jemal Summer's lunch, she then looks at the last lunch in that box. It is Billy Wagner's so she knows she must be close! She looks at the lunch right next to Billy's, and it is hers. Happily, she sees that the cafeteria remembered to pack her a cheese sandwich and carrots.

Characterizing Computational Thinking elements

The steps in computational problem solving can be described globally as follows.

1. decontextualization: translating the problem or question in a subject matter domain ('context') into computational terms;
2. computational problem solving: constructing an executable solution;
3. (re)contextualization: translating the solution back to the subject domain.

Definitions of Computational Thinking vary in the way they emphasize the activities carried out in the steps (1), (2) and (3). Selby and Woollard (2013, p. 5), for example, describe computational thinking as "an activity, often product oriented, associated with, but not limited to, problem solving. It is a cognitive or thought process that reflects

- the ability to think in abstractions,
- the ability to think in terms of decomposition,
- the ability to think algorithmically,
- the ability to think in terms of evaluations,
- the ability to think in generalizations."

Although there is some overlap, we can globally associate the above elements to the steps in our model. The first two elements mainly have to do with Step (1): analyzing patterns within a problem or situation (abstraction), and breaking up a problem into smaller subproblems (decomposition). Algorithmic thinking is used mostly within Step (2), whereas evaluating a solution and investigating how it can be generalized connects the computational solution to the subject matter domain, which takes place in Step (3). We can summarize this in a table:

	(1)	(2)	(3)
Selby & Woollard (2013)	abstraction decomposition	algorithmic thinking	evaluation generalization

Pair Task

You will find three other well-known operationalizations in the document ‘’B. Definitions of CT’’. Extend the above table with three more rows. Categorize the elements you find in the three columns. Do you recognize any elements in the classroom scenarios 1 and 2?

Conclusion

General Discussion Task

Compare your findings with fellow students in class. Together, construct a short working definition of Computational Thinking in terms of steps and activities.

Learning Resources

A. Brief introduction to CT

B. Definitions of CT

References

Selby, C. & Woollard, J. (2013) Computational thinking: the developing definition, available via internet: http://eprints.soton.ac.uk/356481, accessed: 10 February 2021

Yadav, A., Krist, C., Good, J., & Caeli, E. N. (2018). Computational thinking in elementary classrooms: measuring teacher understanding of computational ideas for teaching science. *Computer Science Education*, 28(4), 371–400.

UNIT 2: Computational Thinking Tools

In this unit you will get hands-on experience in computational thinking learning activities using simple yet powerful computational digital tools intended for real-world problem solving applications in different disciplines – no additional programming is required for the moment.

The unit introduces three tools, namely, NetLogo, Microsoft Excel, and Google Ngrams and presents activities that demonstrate the way that these tools can be used to incorporate computational thinking in different school subjects. The description of each tool as well as its applications in different disciplines are provided in detail in the respective sections. As you go through these activities, you engage with computational thinking aspects, practices and tools and understand the role of computational thinking within the different disciplines.

Contribution to the learning outcomes

Learning outcomes
● Engage with and apply computational thinking in practice through a series of case-scenarios examples ● Classify problems as computationally solvable ● Employ computational tools to solve problems ● Develop detailed step-by-step solutions to problems ● Think about, interpret and visualise data ● Use data analysis to enhance understanding of natural systems ● Evaluate what kind of problems can be solved using modelling and simulation. ● Understand and describe how modelling and simulation can be used to solve a problem ● Understand and describe how modelling and simulation can be used to solve a problem ● Analyse data and identify patterns through modelling and simulation. ● Collaborate and communicate with peers to solve a problem

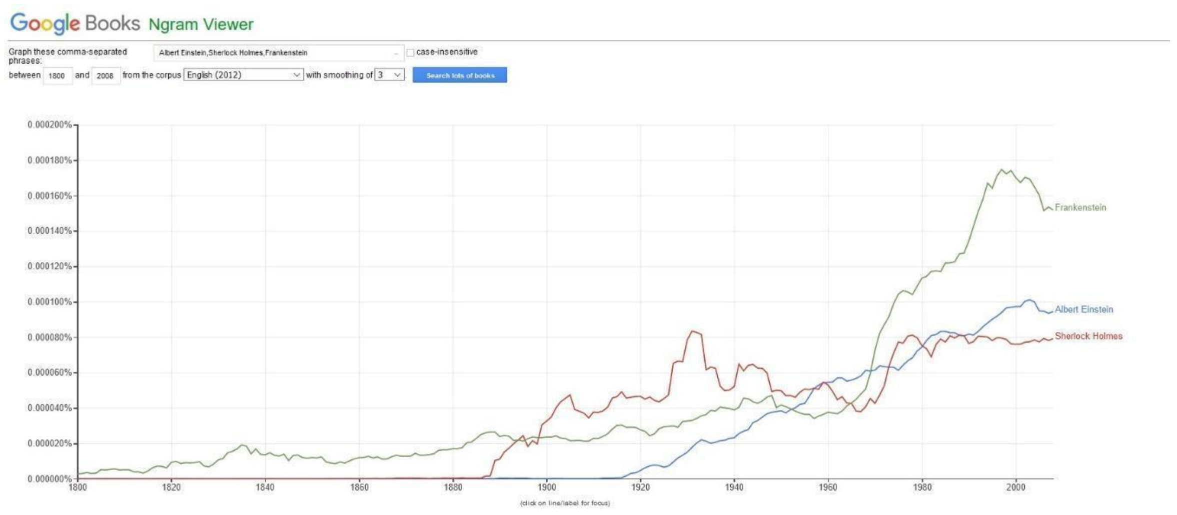
Activity 2.1: Using Google Ngrams

Total time: 2 - 3 hours

The Google Ngram viewer is a graphing-tool that depicts word frequencies from a large corpus of books, visualising a term's or a phrase's use over time. The tool uses literature sources printed between 1500 and 2008 in American English, British English, French, German, Italian, Spanish, Russian, Hebrew and Chinese, and thereby, it can be employed to investigate changes in language over the years. Fluctuations in language use can depict social-cultural changes and thus, provide an understanding of how different social-cultural transformations evolve through history. In this activity, we are going to use Google Ngrams to examine cultural and social changes through time as they are reflected by the use of specific terms in books.

Warm-up discussion

Before you start working in this activity, click on the following link to access the Google Ngram viewer (https://books.google.com/ngrams). You will notice that the tool already suggests an example that is depicted in the following figure. In this example, we see the results for three phrases, «Albert Einstein», «Sherlock Holmes», and «Frankenstein». The tool identifies the frequency of these phrases as they appear in books between 1800-2008. As you can see, the term Frankenstein has been reported in the literature since the early 1800s in comparison with the other two terms which appear later in the literature.


Google Books Ngram Viewer

Graph these comma-separated phrases: Albert Einstein,Sherlock Holmes,Frankenstein ☐ case-insensitive

between 1800 and 2008 from the corpus English (2012) with smoothing of 3

Figure 1 Google Ngram Example for the terms: «Albert Einstein, Sherlock Holmes, and Frankenstein».

Discussion Task

What other trends do you notice regarding the frequency of the terms in figure 1?

Phase 1: Problem statement

The question we will investigate in this activity refers to cultural changes in the United States over the last two centuries. The social-cultural theoretical lens we are going to use are described in the following paragraph followed by our question.

The theory of social change and human development predicts a global shift from *Gemeinschaft* to *Gesellschaft* based on sociodemographic changes. These two words were introduced in 1887 by Toennies and have been used for studying social change. 'Gemeinschaft describes binding, primary interactional relationships based on sentiment and is characterised by a rural environment, simple face to face relationships, low levels of technology, limited education, and of need rather than of wealth. On the other hand, Gesellschaft describes an interactional system characterised by self-interest, competition, and negotiated accommodation and is characterised by an urban environment, a modernised society with high levels of technology and wealth' (source: Younes and Reips, 2018, p.1; Christenson, 1984, p.160)

Within the last centuries, the rate of urbanisation has increased drastically and globally. In this case study, we investigate if this change aligns with a movement from *Gemeinschaft* towards *Gesellschaft* system. Specifically, we will investigate the following question:

«Did the United States move from Gemeinschaft towards Gesellschaft during the last two centuries and does this align with the transition from a rural to an urban society?»

To answer this question, we are going to examine patterns of socio-cultural changes reflected in word-frequencies the last two centuries in the United States. The Google Ngram viewer is particularly useful to this end, as cultural values are reflected in words/terms used in writings, and thus, we can provide evidence for the long-term cultural change in values by examining word frequencies in books.

Phase 2: Selecting search terms

The first step in answering our question is to identify appropriate search terms that are linked to the concepts Gemeinschaft and Gesellschaft and can depict the cultural changes in values in the United States. In this particular example, the following criteria are important for selecting representative search terms: a. a high frequency of the term and b. a narrow range of semantic interpretations.

Discussion Task

Why do you think the above criteria are important in this activity?

*

Selecting terms with high frequency is important so that the graph lines can actually depict cultural changes through time and because words with high frequency are characteristics of a country's culture. Equally important in this example is selecting words with a narrow range of semantic interpretations. That is because words with wide-ranging meanings can be used in different contexts that are not always relevant to cultural values. In the following table, you can see words that fulfil the above criteria and mirror the Gemeinschaft and Gesellschaft system.

Table 1: Terms linked to Gemeinschaft and Gesellschaft direction

Gemeinschaft	obliged, duty, give, benevolence, act, deed
Gesellschaft	choose, decision, get, acquisition, feel, emotion

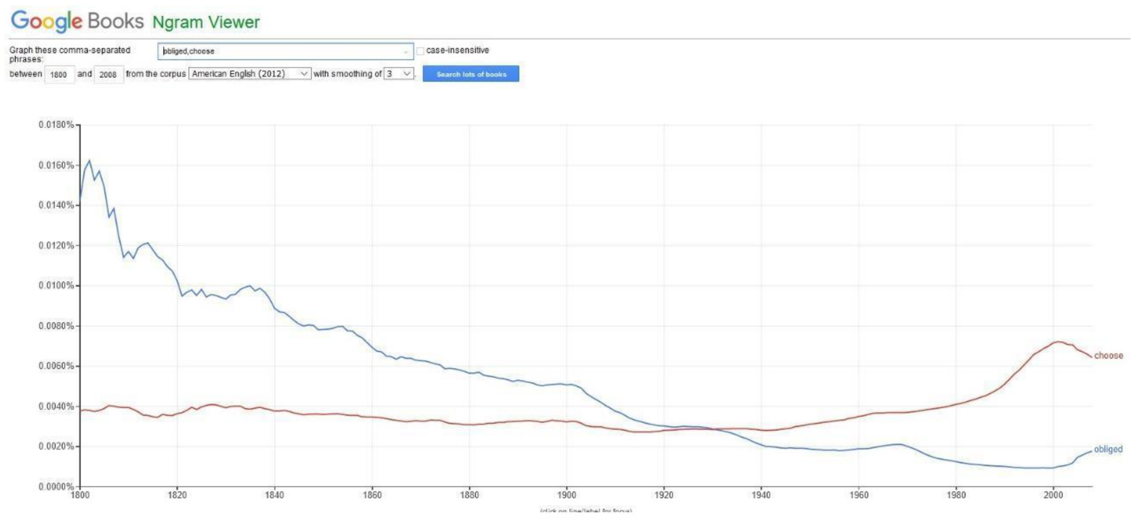
Discussion Task

How can we use the above list and the Google Ngram viewer to answer our question?

We discussed that Google Ngram can be employed to depict frequencies of words reported in the literature through time. For our example, this suggests that we can use and compare terms that reflect opposing attitudes, traits, and characteristics and observe how their frequency fluctuates over time.

Discussion Task

The following figure depicts the frequency of the words obliged (linked with the Gemeinschaft system) and choose (linked with the Gesellschaft system). What do you notice about the frequency of these two terms over time?



Graph these comma-separated phrases: obliged,choose ☐ case-insensitive

between 1800 and 2008 from the corpus American English (2012) with smoothing of 3.

Phase 3: Comparing terms and Interpretation

In this phase, we are going to use Google Ngrams to compare more terms connected to Gemeinschaft and Gesellschaft system and observe how their frequency has changed during the last two centuries.

Group Task

Click on the following link to access Google Ngrams (https://books.google.com/ngrams).

1. Use the search field (erase any search terms already in the field) and type the following two words separated by a comma: Duty, Decision.
2. From the drop-down menu «from the corpus» select American English (2012) and then click the button «search lots of books».
3. What do you notice about the frequency of these two terms over time?
4. In the same way, explore the following frequency of the following words (select at least two):
    a. Give and Get,
    b. Benevolence and Acquisition,
    c. Act and Feel,
    d. Deed and Emotion
5. How do the frequencies of these words fluctuate?

Do your observations verify the transition from Gemeinschaft towards Gesellschaft during the last two centuries?

*

Indeed, as predicted by the theory of social change and human development, terms like obliged, duty, give, benevolence, act, and deed, all of which reflect a rural and a Gemeinschaft environment, weakened over the last two centuries. In this same period, terms like choice, decision, get, acquisition, feel, and emotion, all of which reflect an urban and a Gesellschaft environment, increased over this period.

Generalisation (optional)

In this activity, it was evident that, as the United States moved in the Gesellschaft direction, Gesellschaft cultural features (reflected by relevant words in the corpus of American books) showed a quantitative increase, whereas Gemeinschaft cultural features (reflected by relevant words in the corpus of American books) showed a quantitative decrease. Can these relationships and trends be generalized to other parts of the world as the theory of social change and human development would predict?

Group Task

Repeat the steps described in Phase 3; this time select the literature of your choice (e.g., British or German) to answer the same question for the country of your choice.

Note: For German literature you can use the following terms (source: Younes & Reips, 2018):

1. Versprechen and auswählen
2. Pflicht and Entscheidung
3. Geben and bekommen
4. Güte and Kauf
5. Handeln and spüren
6. Handlung and Emotion

Learning Resources

A Gemeinschaft to Gesellschaft.pdf (optional)

B The changing psychology of culture from 1800 through 2000 (optional)

C The changing psychology of culture in German‑speaking countries (optional)

References

Christenson, J.A., (1984). Gemeinschaft and gesellschaft: Testing the spatial and communal hypotheses. *Social Forces*, 63(1), pp.160-168.

Greenfield, P.M., (2013). The changing psychology of culture from 1800 through 2000. *Psychological Science*, 24(9), pp.1722-1731.

Younes, N. and Reips, U.D., (2018). The changing psychology of culture in German‐speaking countries: A Google Ngram study. *International Journal of Psychology*, 53, pp.53-62.

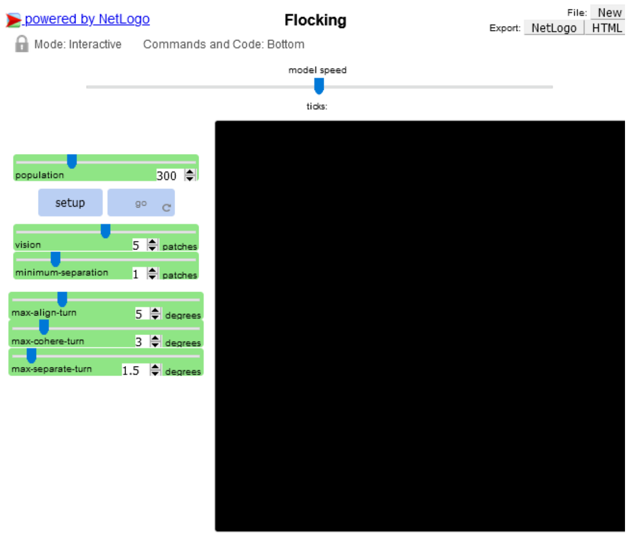
Activity 2.2 Modelling and Simulation through NetLogo

Total time: 2.30 – 3 hours

NetLogo is a multi-agent programmable modelling environment for simulating natural and social phenomena and demonstrate how these develop over time. With this tool, you can create a world made up of rectangles or patches and parameterise agents (or turtles) that move around and interact with each other and their environment. In this activity, we are going to use NetLogo on a case study about epidemiology and run model simulations to investigate how large and small changes can affect an environment.

Warm-up discussion

Before you start working in this activity, click on the following link to access the tool http://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Art/Fireworks.nlogo. From the drop-down menu «Search the Models Library» select the option sample models/Biology/Flocking. The example that is loaded is depicted in the following figure.



population 300
vision 5 patches
minimum-separation 1 patches
max-align-turn 5 degrees
max-cohere-turn 3 degrees
max-separate-turn 1.5 degrees

Figure 1 Flocking example in NetLogo

This model is an attempt to mimic the flocking of birds. Each bird is following exactly the same set of rules: "alignment", "separation", and "cohesion". "Alignment", controlled by the max-align-turn slider, means that a bird tends to turn so that it is moving in the same direction that nearby birds are moving. "Separation", controlled by the max-separation-turn slider, means that a bird will turn to avoid another bird which gets too close. "Cohesion", controlled by the max-coherence-turn slider, means that a bird will move towards other nearby birds (unless another bird is too close). When two birds are too close, the "separation" rule overrides the other two, which are deactivated until the minimum separation is achieved. The vision slider is the distance that each bird can see 360 degrees around it.

The green range sliders (e.g. max-fireworks) on the left are used to parameterise your model. The button SETUP, sets up the model according to the values indicated by all the sliders. The button GO executes the model.

Pair Task

First, determine the number of birds in the simulation and set the population slider to that value. Press first SETUP, and then GO to start the simulation (The default settings for the sliders will produce reasonably good flocking behaviour. However, you can play with them to get variations). Adjust the sliders to see how you can get tighter flocks, looser flocks, fewer flocks, more flocks.

Phase 1: Problem statement

The question we will investigate in this activity refers to epidemiology and particularly to the Zika virus.

Zika virus is a mosquito-borne flavivirus that was first identified in Uganda in 1947 in monkeys. The incubation period (the time from exposure to symptoms) of the virus disease is estimated to be 3–14 days. The majority of people infected with the Zika virus do not develop symptoms. Symptoms are generally mild including fever, rash, conjunctivitis, muscle and joint pain, malaise, and headache, and usually last for 2–7 days. The virus is primarily transmitted by the bite of an infected mosquito from the *Aedes* genus, mainly *Aedes aegypti*, in tropical and subtropical regions. Aedes mosquitoes usually bite during the day, peaking during early morning and late afternoon/evening. This is the same mosquito that transmits dengue, chikungunya, and yellow fever. The virus is also transmitted from mother to fetus during pregnancy, through sexual contact, transfusion of blood and blood products, and organ transplantation (source: World Health Organisation[1]).

In this case study, we are going to work on a hypothetical epidemiological scenario and try to understand and assess the risk of the virus spread. The scenario we are going to work on is the following:

In a small city in South Africa, two people have been tested positive with the Zika virus. The government wants to predict how serious the situation is and thus take the necessary prevention

[1] https://www.who.int/news-room/fact-sheets/detail/zika-virus

measures. The total population in this city is 800 and after an initial estimation, the number of mosquitoes in the area is 1040 while the number of predators in the area is 21.

The question that we need to answer is the following:

How many people are going to be infected and what would be the best strategy for the virus prevention?

To answer this question, we are going to use NetLogo and run simulations on a model about the Zika virus.

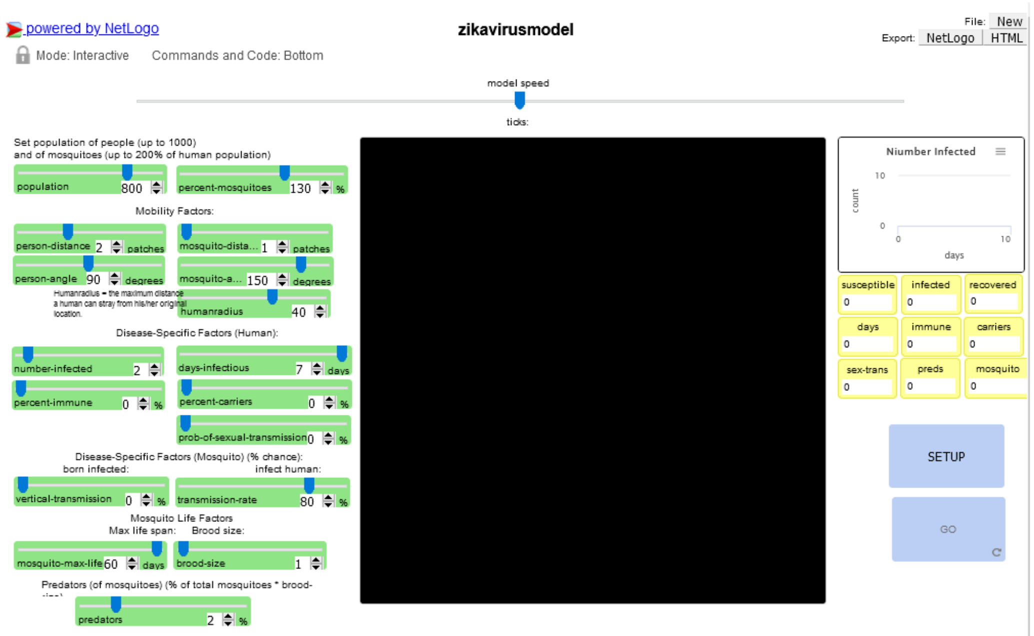
Phase 2: Simulation

In this phase, we are going to work with an already implemented model that will help us understand how modelling and simulation can be employed to generate new understandings and knowledge about a phenomenon and how they can affect decision making.

Click on the following link to access NetLogo:

http://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Art/Fireworks.nlogo

Click on «browse» to upload the model zikavirusmodel.nlogo. Once the model loads successfully, the following example should be evident.


powered by NetLogo

Mode: Interactive   Commands and Code: Bottom

zikavirusmodel

File: New
Export: NetLogo | HTML

model speed

ticks:

Set population of people (up to 1000)
and of mosquitoes (up to 200% of human population)

population 800
percent-mosquitoes 130 %

Mobility Factors:

person-distance 2 patches
mosquito-dista... 1 patches
person-angle 90 degrees
mosquito-a... 150 degrees

Humanradius = the maximum distance a human can stray from his/her original location.

humanradius 40

Disease-Specific Factors (Human):

number-infected 2
days-infectious 7 days
percent-immune 0 %
percent-carriers 0 %
prob-of-sexual-transmission 0 %

Disease-Specific Factors (Mosquito) (% chance):
born infected: vertical-transmission 0 %
infect human: transmission-rate 80 %

Mosquito Life Factors
Max life span: mosquito-max-life 60 days
Brood size: brood-size 1

Predators (of mosquitoes) (% of total mosquitoes * brood-size)

predators 2 %

susceptible	infected	recovered
0	0	0
days	**immune**	**carriers**
0	0	0
sex-trans	**preds**	**mosquito**
0	0	0

Figure 2 Zika virus in NetLogo

In the model, mosquitoes would bite Zika-infected humans and then go on to spread the disease by biting other, non-infected humans. There are many variables that you can manipulate in this model. For this activity, we are going to focus only on the following:

- *person-dist*ance which represents the mobility of humans - how far they can turn and how many squares (patches) they can travel in a day
- *number-infected* which represents the number of initially infected people
- *days-infectious* which represents the length of time a typical person is infected
- *percent-immune* refers to the percentage of the population that is immune (= vaccinated)
- *predators* which represent the number of predators
- *percent-mosquitoes* which represent the number of mosquitoes

For every run of the model, you must do the following three things in sequence: first, set the sliders (in case you want to adjust them), secondly, click SETUP (in the lower right corner of the display) and third, click GO to run the model.

The plot in the upper right corner and the monitors right below it, tell you how the society is faring with respect to the virus under your slider regime.

Pair Task

Run the simulator with the default values which is in line with the problem statement presented in phase 1. As soon as the simulation ends, answer the following question: How many people were infected in total and in how many days?

Running the simulator will give you an estimation of how many people will get infected with the virus under specific circumstances. In this example, this number lies somewhere between 740 and 775. Therefore, using our model, we were able to answer one part of our question regarding the number of people that would get infected from the Zika virus. What we haven't yet answered is what strategy would be best to prevent the spread. This is the focus of the next phase.

Phase 3: Further investigation - Interpretation

There are several variables in this model that can be adjusted and explore how the disease would spread among the population. For investigating the best strategy to prevent the spread, we are going to consider three variables: predators, person-distance, and percent of immune.

Group Task

In this task, we are going to investigate how the following changes affect the spread of the virus and which one is more effective.

1. Increase predators by 6%: Change the predators' value to 6%, and then click on SETUP and then GO. Keep a note on the number of recovered in the monitors in the upper right corner. Before you continue to number 2 below, set the predator slider back to 2% (default value).

2. Increase the number of immune to 6%: Change the value of immune to 6%, and then click on SETUP and then GO. Keep a note on the number of recovered in the monitors in the upper right corner. Before you continue to number 3 below, set the immune slider back to 0 (default value).
3. Decrease person movement to 1 patch: Change the value of person-distance to 1%, and then click on SETUP and then GO. Keep a note on the number of recovered in the monitors in the upper right corner.

*

You will notice that if you decrease the person movement by 1 patch, the number of infected people lie somewhere between 400-680. If you increase predators by 6%, then the number of infected people lies between 140 – 300 while by increasing the percentage of immune to 6%, the number of infected people lies between 630 – 740. Therefore, under the current conditions that our model considers, the best strategy to prevent the spread to the whole population is to increase the predators by 6%.

Phase 4: Wrap-up question

In this activity, we worked on a hypothetical scenario in epidemiology and by using modelling and simulation, we examined the dynamics of a virus spread by manipulating variables and evaluating the results.

Discussion Task

How can modelling and simulation influence decision making in other disciplines other than epidemiology?

Learning Resources

zikavirusmodel.nlogo

Activity 2.3 Using Microsoft Excel to manipulate and represent data

Total time: 2.30 – 3 hours

Excel is a spreadsheet tool for organising, performing calculations on data, and analyse and represent data as a chart or a graph. In this activity, we are going to use Excel for analysing data and representing data, identifying trends depicted in the graphs and relate information to answer a question about climate differences in biomes and how these affect their surface colours.

Warm up - discussion

Before you start working in this activity, open the Earth colour workbook and navigate to the Mapping biomes worksheet to reveal a picture of the Earth's surfaces created from satellite images (figure 1).

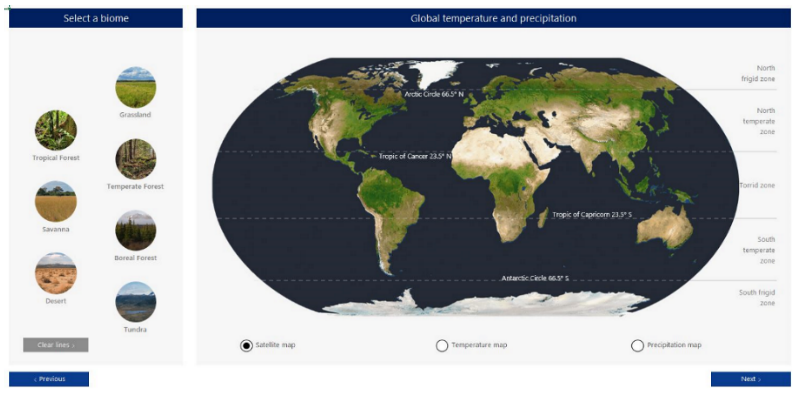


Figure 1 Mapping Biomes worksheet

Pair Task

Click on the biome buttons on the left of the earth's satellite image to highlight the locations of different biome regions.

- What colour do you think best describes each biome?
- What accounts for the different colours related to different biomes?

Tip: select the Temperature and Precipitation map button located below the satellite image to make connections between the colours and physical conditions.

*

As you notice, regions with a lot of precipitation are greener while higher temperatures and lower precipitation correspond to regions that may appear browner. The poles have very low average temperatures accounting for the white colour related to snow and ice.

Phase 1: Problem Statement

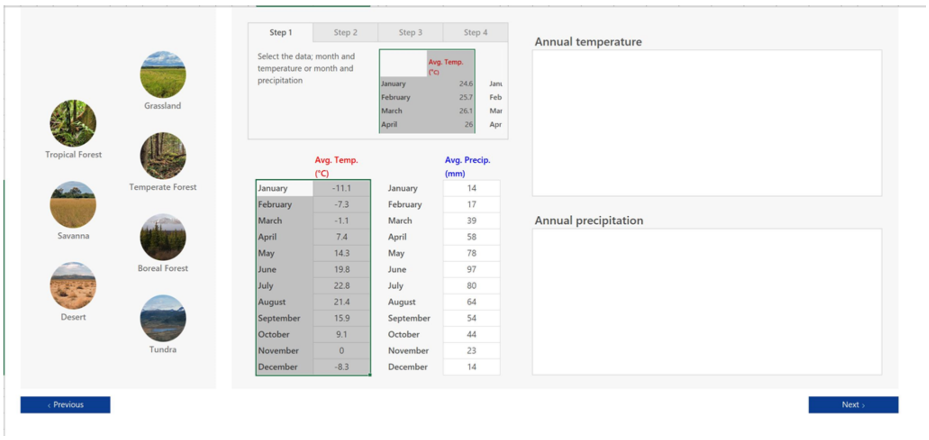
In this activity, we are going to use data as a driver to better understand why changes in colour occur in biomes. We will investigate the seven terrestrial biomes and their characteristics colours seen from space which are related to factors such as temperature and precipitation. The question we will address is the following:

How do the temperature and precipitation in different biomes change through the year and how is this change reflected through colours?

Phase 2: Graphing Temperature and Precipitation

In the warm-up task, you looked at average global temperature and precipitation data and you started making connections between the colours of Earth's surfaces depicted in the Satellite image and the average yearly precipitation and temperature. In this phase, we are going to investigate monthly changes in precipitation and temperature happening through the year in biomes.

For the following task, navigate to the «Create a biome climate chart» worksheet depicted in the following figure.



Select the data; month and temperature or month and precipitation

	Avg. Temp. (°C)
January	24.6
February	25.7
March	26.1
April	26

	Avg. Temp. (°C)		Avg. Precip. (mm)
January	-11.1	January	14
February	-7.3	February	17
March	-1.1	March	39
April	7.4	April	58
May	14.3	May	78
June	19.8	June	97
July	22.8	July	80
August	21.4	August	64
September	15.9	September	54
October	9.1	October	44
November	0	November	23
December	-8.3	December	14

Figure 2 Create a biome climate chart worksheet

Pair Task

For this task, select one biome that you would like to investigate further. If you click on the biome's buttons on the left, you will see its monthly temperature and precipitation data being depicted under the «Avg Temp» and «Avg. Precip» labels.

1. Select the month temperature data and then click on the insert option in Excel's menu bar. Click on the recommended charts option and then select a chart you would like to use for depicting the biome's monthly temperature data.
2. Repeat the same steps for the precipitation data

How does the temperature and precipitation data fluctuate over the year for your biome? Observe which month(s) has the highest and lowest temperature and precipitation.

Select another biome and repeat this task by creating a graph to depict its monthly temperature and precipitation. Compare the two biomes by observing which one shows more or fewer fluctuations in temperature and precipitation through the year.

Optional task

Repeat the above task for all biomes and select different types of charts. Observe which charts are more useful for depicting the changes in temperature and precipitation and for discerning trends, differences and similarities between two or more biomes.

*

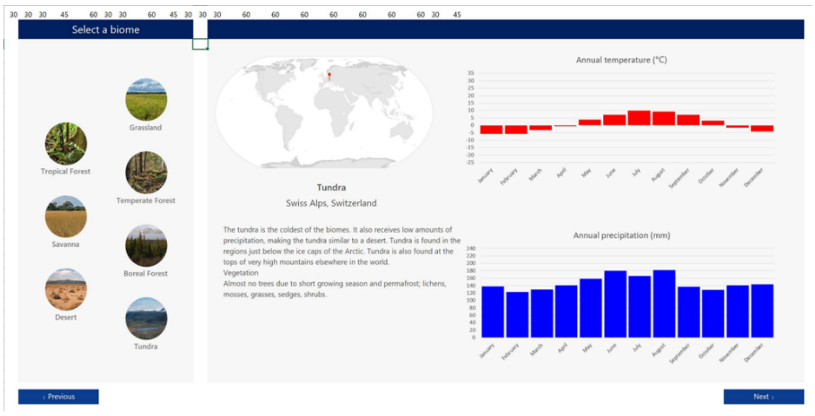
At this point, we have addressed the first part of our question regarding the way temperature and precipitation change over a year for a biome. By using graphs to depict the temperature and precipitation data for a biome, we were able to observe these fluctuations and also to compare biomes with each other. What we need to further investigate, however, is how these changes in temperature and precipitation data are reflected through colours.

Phase 3: Comparing colours with temperature and precipitation in biomes

In the previous phase, we used graphs to depict and compare monthly changes in temperature and precipitation data for two biomes. In this phase, we are going to correlate biome characteristics, temperature and precipitation data and investigate how changes in seasonal vegetation are reflected by the most prominent colour seen in a biome's landscape for a given month of the year.

Pair Task

Navigate to the «Information on biomes» worksheet depicted in the following figure. Select again the biomes you chose in the previous phase and read the information provided regarding their vegetation. What do you notice regarding a biome’s temperature, precipitation and vegetation?



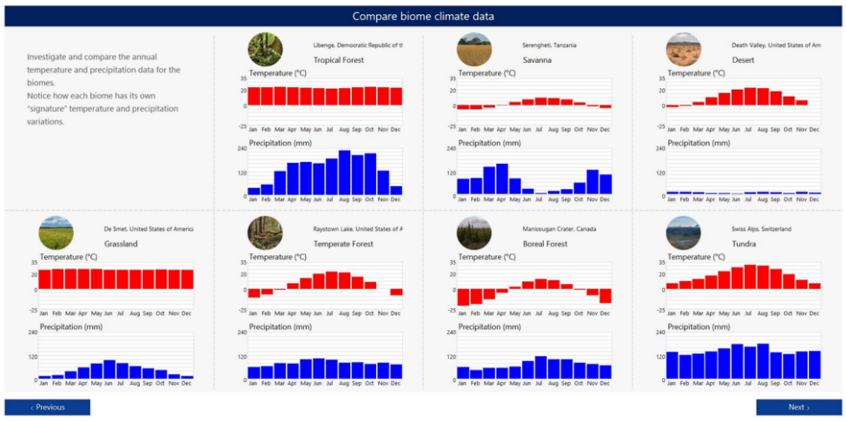
Tundra
Swiss Alps, Switzerland

The tundra is the coldest of the biomes. It also receives low amounts of precipitation, making the tundra similar to a desert. Tundra is found in the regions just below the ice caps of the Arctic. Tundra is also found at the tops of very high mountains elsewhere in the world.
Vegetation
Almost no trees due to short growing season and permafrost; lichens, mosses, grasses, sedges, shrubs.

Figure 3 Information on biomes worksheet

What you will notice by comparing a biome's temperature, precipitation and vegetation, is that the vegetation of a particular region is specially suited to the climate conditions of the biome region. You would also notice that even if temperatures of one region may be similar to another, differences in precipitation have a big impact on the type of vegetation in a biome and therefore, its surface colours.

For the following question, navigate to the «*Compare biome climate data*» worksheet depicted in the following figure.



Investigate and compare the annual temperature and precipitation data for the biomes.
Notice how each biome has its own "signature" temperature and precipitation variations.

Figure 4 Compare biome climate data worksheet

Pair Task

Based on the temperature and precipitation data,

- Which biome regions do you predict to have the most consistent yearly surface colours?
- Which do you predict to have the greatest fluctuation in surface colours during the calendar year?

Explain your reasoning.

*

Colour can be used to describe biome changes throughout the year; the changing vegetation of a biome accounts for its surface colour and is reflected by the most prominent colour seen in the biome's landscape for a given month of the year.

For the following question, navigate to the «Biome colours through the year» worksheet depicted in the following figure.

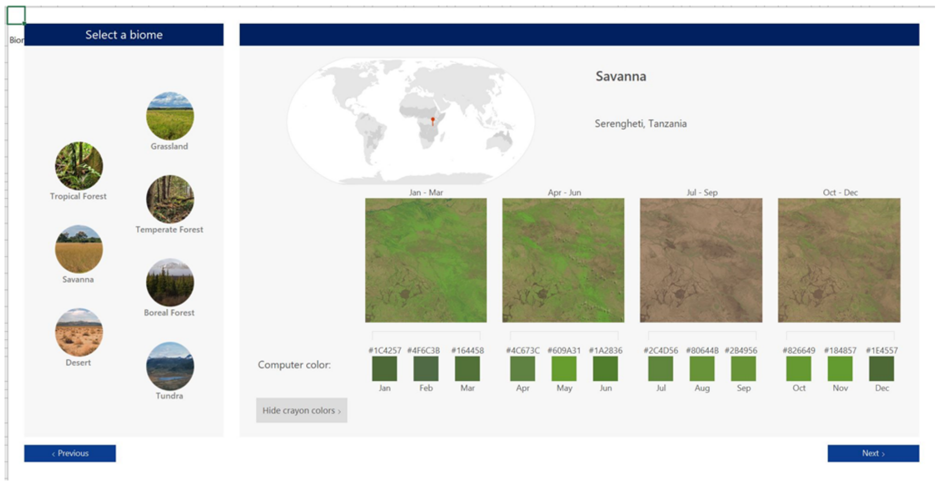


Figure 5 Biome colours through the year worksheet

Pair Task

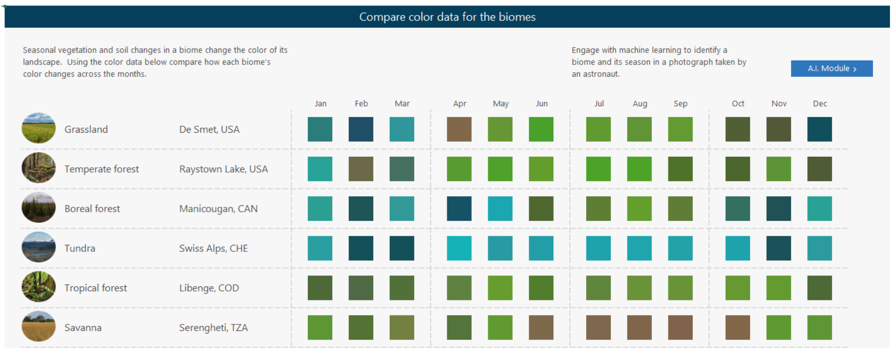
Based on the temperature and precipitation data,

- Which biome regions do you predict to have the most consistent yearly surface colours?
- Which do you predict to have the greatest fluctuation in surface colours during the calendar year?

Explain your reasoning.

*

For the following question, navigate to the « Compare biome colour data» worksheet depicted in the following figure.



Compare color data for the biomes

Seasonal vegetation and soil changes in a biome change the color of its landscape. Using the color data below compare how each biome's color changes across the months.

Engage with machine learning to identify a biome and its season in a photograph taken by an astronaut.

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Grassland	De Smet, USA	🟦	🟦	🟦	🟫	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦
Temperate forest	Raystown Lake, USA	🟦	🟫	🟦	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Boreal forest	Manicougan, CAN	🟦	🟦	🟦	🟦	🟦	🟩	🟩	🟩	🟩	🟦	🟦	🟦
Tundra	Swiss Alps, CHE	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Tropical forest	Libenge, COD	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Savanna	Serengheti, TZA	🟩	🟩	🟩	🟩	🟩	🟫	🟫	🟫	🟫	🟫	🟩	🟩

Figure 6 Compare biome colour data worksheet

Pair Task

How are the monthly signature colours for the different biome regions related to temperature and precipitation? Provide examples from the data.

Tip: Notice how the different colours in different biomes are related to precipitation and temperature.

*

You may have noticed that the teal colour represents areas with snow, while areas with a lot of precipitation are greener, and higher temperatures and lower precipitation correspond to regions that may appear browner.

Answering the second part of our question - *how changes in precipitation and temperature are reflected in colours* - we observed that the changing vegetation of a biome, as resulted by changes in temperature and precipitation, is reflected by the most prominent colour seen in the biome's landscape for a given month of the year, and thus, accounts for the biome’s surface colour.

60

Phase 4: Predicting Climate change - Extended activity (optional)

The tasks and questions in the previous phases guided you to answer the question about how the temperature and precipitation in different biomes change through the year and how this change is reflected through colours. In this phase, you are going to examine how a climate change concern impacts a biome region of your choice. To this end, you are going to collect and analyse precipitation and temperature data for this biome and investigate how its signature colours might change if the climate change concern is left unchecked.

Group Task

Research a climate change concern that impacts a biome region of your choice.

1. Collect monthly temperature and precipitation data from the past 10 years to see if there are any trends or interesting patterns. Record the data in your excel workgroup in an appropriate format.
2. Create temperature and precipitation graphs for the biome region to help you identify trends.
3. Think about the climate change concern for your biome region and how it is reflected in the data you collected. Discuss how the colours in this biome region might have changed as a result of the climate change concern and make predictions about how the colours in that region might change over the next 10 years if the climate change concern is left unchecked.

Learning Resources

Workbooks: Earth colour workbook.xlsx

UNIT 3: Experience Computational Thinking via programming

Total time: 10 hours

In this unit you will get acquainted with the basics of programming and will practice algorithmic thinking in the context of storytelling and games. You will also learn to use flowcharts to represent algorithms.

Contribution to the learning outcomes

Learning outcomes
● generate and write down ideas for a story in a standardized way ● identify the key differences in developing electronic interactive stories and traditional paper-based books ● identify and explain the algorithm for an existing program ● make modifications to an existing program ● debug and correct an existing program ● plan and design a new program to produce an interactive story ● create and develop an interactive story using programmable elements ● use loops, variables, broadcast messages, IF statements and sequential instructions within a program ● understand the importance of correct instructions ● understand what an algorithm is ● represent an algorithm in a flowchart ● evaluate the effectiveness of an algorithm ● implement a pre-written algorithm or flowchart within Scratch ● explain what is meant by the term variable ● create and use variables within your program

- explain what is meant by the term ‘selection’ and ‘loop’
- use selection and loop statements within an algorithm or program

Activity 3.1: Storytelling with Scratch

Digital storytelling uses digital media (images, voice, music, motion) to tell a story. Over the past few years, digital storytelling has become an increasingly popular and effective learning activity to meet a range of learning goals, in particular the learning objectives related to computational thinking.

In this unit you will practice digital storytelling in Scratch. Scratch is a free educational, visual and block-based programming environment. In Scratch, students can develop their ideas in the form of projects using programmable media such as videos, images, games, and animations.

Pair Task

Watch the video the Intro to Scratch

View this sample project created by eighth-graders studying the periodic table.

*

With Scratch, you

- Formulate a problem to determine how to use the elements in Scratch to construct your story -- creating plot, setting, sequencing, and perspective.
- Logically organize and analyse data by creating blocks of code to create characters and their environment.
- Represent story content through the movement of sprites -- the characters in Scratch.
- Use algorithmic thinking to develop code that makes sprites move and communicate.

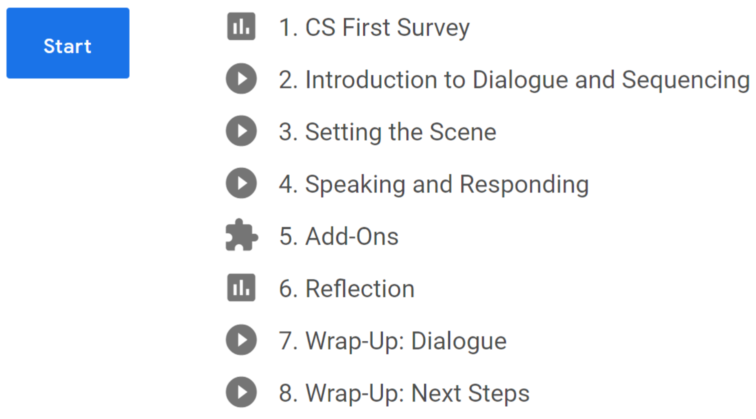
Preparation: Get things ready

We assume that your instructor has provided you with CS First usernames and passwords. Before you can get started, do the following:

- Open a new window in your browser and go to g.co/CSFirst
- Click "Sign in" in the top right
- Click "I am a student"
- Click "Sign in with CS First"
- Click "Enter class code"
- Enter class code 3h24s3
- Enter your Username and Password

Dialogue

Learn about CS First and then create a story in which two characters talk without using questions.


Start

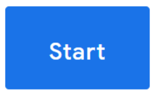
1. CS First Survey
2. Introduction to Dialogue and Sequencing
3. Setting the Scene
4. Speaking and Responding
5. Add-Ons
6. Reflection
7. Wrap-Up: Dialogue
8. Wrap-Up: Next Steps

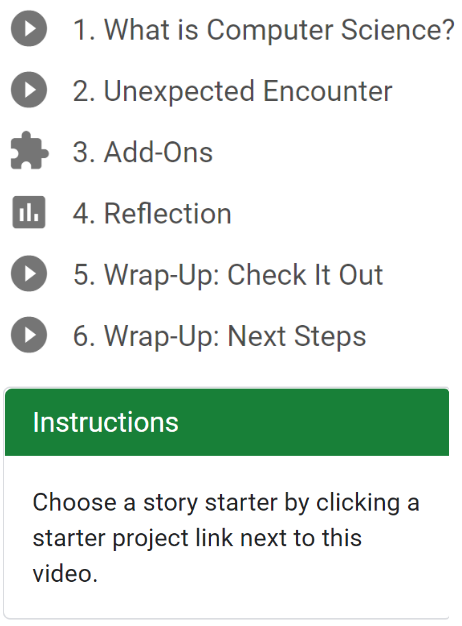
Pair Task

- Press Start button.
- Skip the survey and choose 2: Introduction to Dialogue and Sequencing
- Add the following tab to your browser window: https://scratch.mit.edu/
- If you don't have a Scratch account yet, register yourself.
    - Hint: You can quickly switch between tabs using ctrl tab
- Start video at 2:30, follow the instructions at the end.
- Press next
- Setting the Scene (3)
    - At this point you should have created a new scratch project, say with name myFirstStory
- Start video and follow the instructions at the end: you don't have to interrupt the video in between.
- Press next
    - Speaking and Responding (4)
- Start video and follow the instructions at the end.
- Press next
    - Add-Ons (5)
- Choose 'Adding Motion' at the bottom of the page.
    - Hint: you don't have to enter coordinates yourself if you first place the character at its desired position before you select the corresponding move block.
- Choose 'Add a Third Character'.

Check it out

Tell a story where a character walks through a scene describing what they see.


Start


1. What is Computer Science?
2. Unexpected Encounter
3. Add-Ons
4. Reflection
5. Wrap-Up: Check It Out
6. Wrap-Up: Next Steps

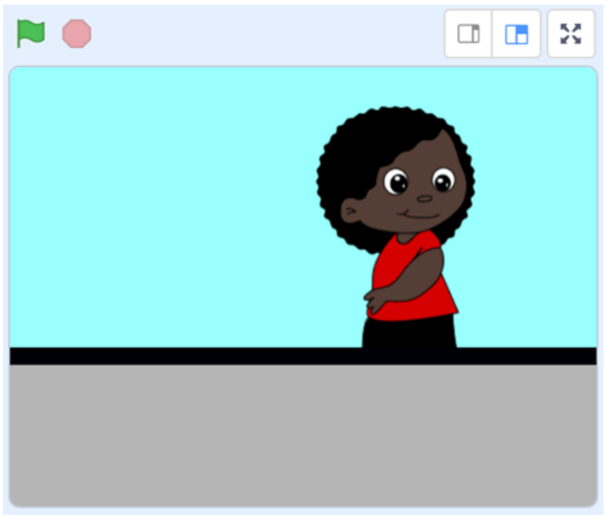
Instructions

Choose a story starter by clicking a starter project link next to this video.

Pair task

- Press Start button.
    - You don't have to watch the first video, but instead you should click on the link

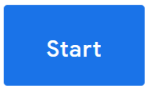
with Starter Project 1. It will open Scratch with a starter project that already includes some code.

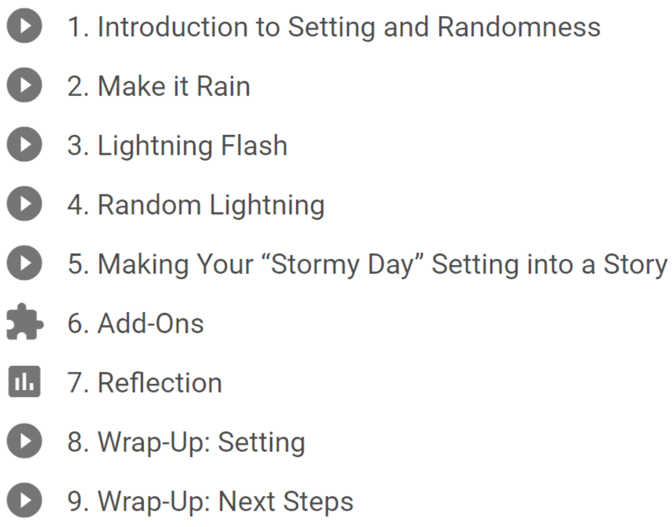


- Press next
    - Unexpected Encounter (2)
- Follow the instruction in the video
- Press next
    - Add-Ons (3)
- Choose ‘Add Sound’ and follow the instructions on the video.

Setting

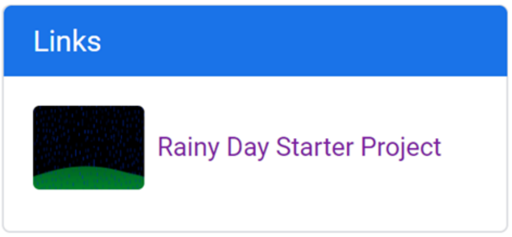
Create a dynamic stormy day setting, complete with rain and lightning.




1. Introduction to Setting and Randomness
2. Make it Rain
3. Lightning Flash
4. Random Lightning
5. Making Your “Stormy Day” Setting into a Story
6. Add-Ons
7. Reflection
8. Wrap-Up: Setting
9. Wrap-Up: Next Steps

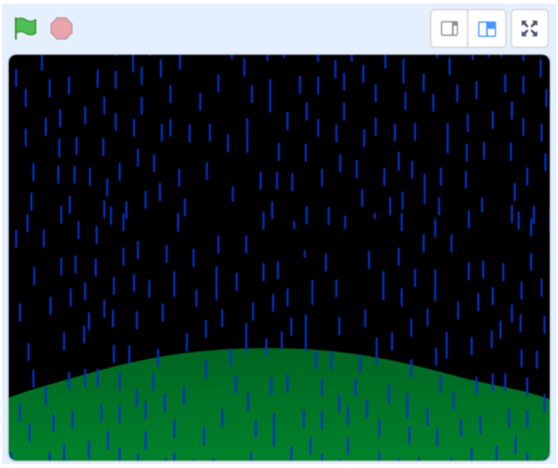
Pair task

- Press the Start button.
- Open the Rainy Day Starter Project


Links

Rainy Day Starter Project

- Go to
    - Introduction to Setting and Randomness (1)



- Watch the video
- Press next
    - Make it Rain (2)
- The video explains how to add rain to your scenario. This is done by using a special single rain sprite, containing all the raindrops, that is programmed to move from top to bottom.
- Follow the instructions in the video
- Press next
    - Lightning Flash (3)
- Follow the instructions in the video
- Press next
    - Random Lightning (4)
- Follow the instructions in the video
- Go to
    - Add-Ons (6)
- Watch the short video, and choose one or two of the possible add-ons at the bottom of the page.

Activity 3.2 Creating a maze game

Introduction: Algorithms

An algorithm is a list of instructions: a stepwise plan. If an algorithm has been formulated precisely enough, someone else should be able to follow the steps exactly as you meant it. In particular, algorithms that are designed to be carried out by a computer are formulated in a very precise language, program code, which tells the computer exactly, step by step, what it should do.

Pair task

Watch the following video: [What is an algorithm and why should you care?]

Phase 1: Exploring algorithms using flowcharts

Consider the following algorithm.

1. Draw a vertical line
2. Draw a horizontal line across it
3. Draw a diagonal line from the top of the vertical to the tip of the horizontal line
4. Repeat instruction 3 for all remaining corners

5. Draw a wavy line from the bottom tip

Group task

Draw a picture from these instructions.

Compare your drawing with that of others. Discuss why it was or wasn't easy and explain why.

*

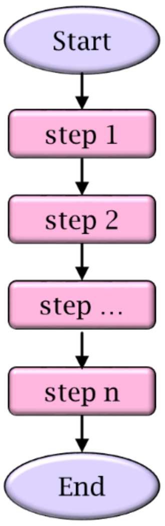
The intended result of the algorithm was a drawing of a kite (see appendix A for a possible representation). The example shows that when the instructions are unclear or incomplete, the result can be ambiguous. Sometimes the performer is expected to resolve any ambiguities himself. Think about what would happen if we always followed instructions exactly as they are given.

Pair task

Watch the video the exact instructions challenge.

*

If we want to describe an algorithm, we have to agree on a language that we use for this. Such a language must be sufficiently precise: it is certain what to do if we follow an algorithm written in that language. An algorithm can be visually represented in a flowchart. This helps maintain overview, which in turn makes it easier to write, read and analyse instructions.



The simplest algorithms consist of a series of instructions that are executed one after the other. In a flowchart we display such a sequence of instructions as shown on the right. You begin at 'Start' at the top. The part between the 'Start' and 'End' (the body of the flowchart) describes what it actually does.

Self-practice task

Drawing a square

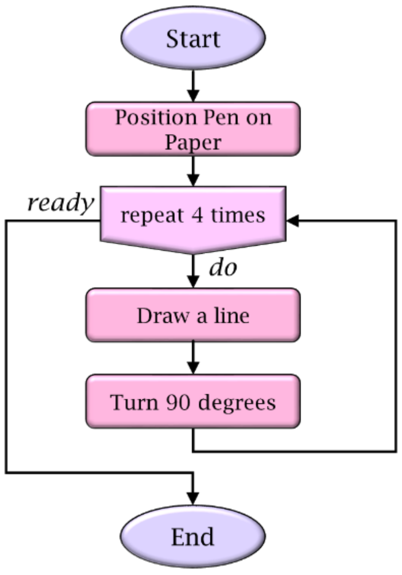
Let's assume you can use the following basic commands:

- **Draw a line**
- **Turn 90 degrees**
- **Position pen on paper**

Choose appropriate instructions and position these in a flowchart to draw a square.

*

Your solution probably consists of a long series of instructions that contains repetitions of code. We can write down such a series more compactly and therefore more clearly by using a repeat/while construct: as long as we have not yet performed 4 repetitions, the do-branch is followed. After the fourth iteration we follow the ready arrow and the algorithm ends.

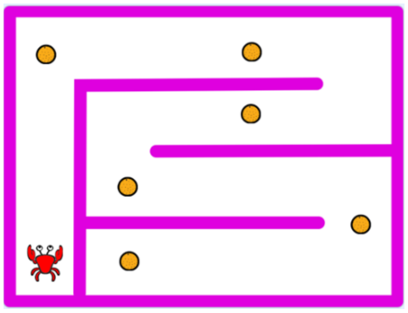


Pair Task

- one student chooses a picture and writes instructions to explain to their partner how to draw the picture
- agree on a set of basic commands in advance that may be used in the description
- the other student follows these instructions meticulously
- discuss the solution and correct mistakes

Phase 2: Create a maze game using sprites

In this phase, you will design a game in Scratch in which the player controls a crab using the arrow keys. The aim of the game is to find your way to the other side of the maze without touching the walls. In the maze, oranges are scattered. The player collects points by making the crab pick up the oranges.



Learning Resources

Scratch file: crab-in-maze-start.sb3

Pair task

Try to work out how your game might run. Make a rough outline of your game. Think about how the game looks and how the main character is controlled by the player. Try to formulate conditions that indicate whether the game has ended. What 'treasures' do you add to your game? Where are they located and how does the player pick them up?

*

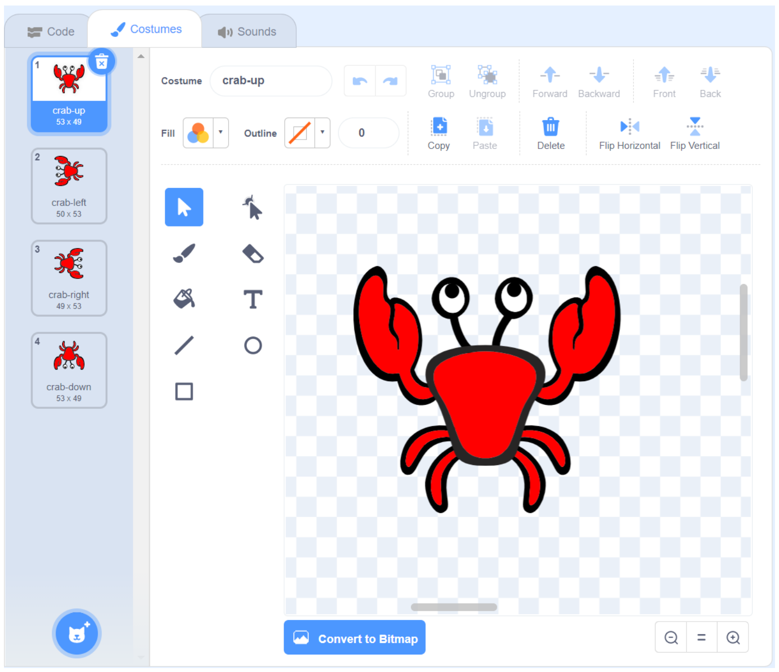
In order to program this game, you will learn how to

- define sprite costumes, positioning and movement
- handle keyboard input
- handle sprite interaction with the environment
- program sprites interacting with each other
- use variables to maintain a score

We'll start by tweaking the look of the main character from the game: the crab (implemented in Scratch as a sprite).

Pair Task

- Go to Scratch and open the file crab-in-maze-start.sb3 (by choosing 'load from your computer' from the File menu).
- Select the crab sprite and choose the Costumes tab.
- Remove the second image, choose the first image, and make the claws of the crab a bit smaller and scale the whole picture so that the crab fits comfortably in the maze (so the walls are not touched)
- Duplicate the image 3 times (by right-clicking on the image and selecting 'duplicate') and rotate these copies for a bird's eye view of your crab: left, right, up and down.

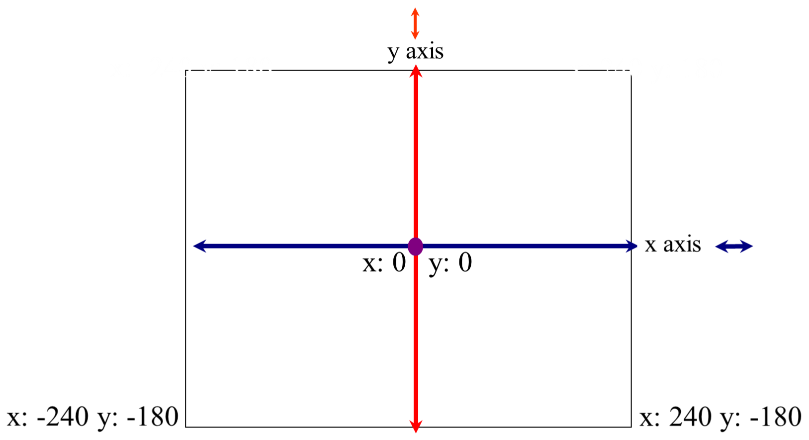


- Change the names accordingly.

*

To program the movement of the crab it is necessary to know how sprites are positioned in Scratch. Scratch represents each point in the playing field as a pair of two numbers: the first indicates the horizontal distance from the middle of the field, the second the vertical distance.

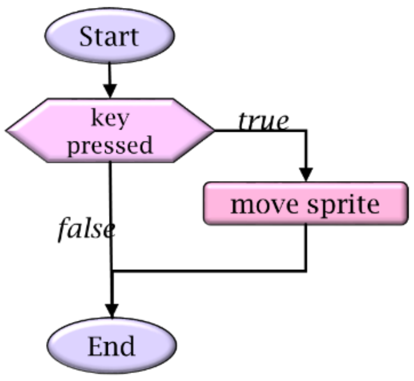
In more technical terms, Scratch uses a so-called coordinate system with the origin in the middle of the playing field. Each point is represented by a pair (x, y) where x is the horizontal distance from the origin and y is the vertical distance.



The total playing field is 480 pixels wide and 360 pixels high. The upper-left corner and upper-right corner have coordinates (-240,180) and (240,180), respectively.

Back to our game. The player moves the crab using the arrow keys. The question now is how we can detect in our program whether the player has pressed a key and how we can then adjust the position of the crab according to the key pressed,

The algorithm that handles the handling of the pressed arrow keys can be specified in a flow chart as follows:

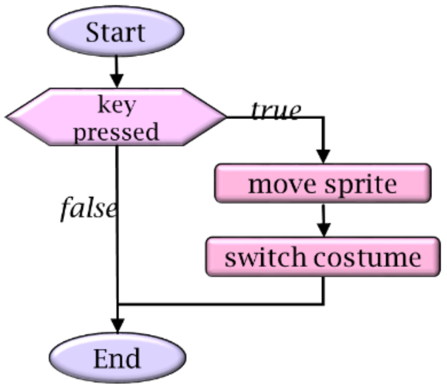


We used a selection here, also known as an if-then construct. In such a selection, it is determined on the basis of a conditional expression (indicated in the hexagonal block) whether the then-branch (the branch with the label true) should be executed. The latter only happens if the condition is true.

Pair task

- Try to find out which *event block* you can use to respond to pressed arrow keys
- Adjust the position of the crab correctly. Which *motion block* do you need?

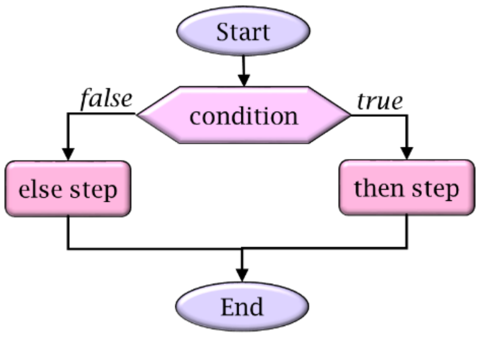
  To make the game more realistic, we will make the picture of the crab match the direction of its movement. We do this by always selecting the right costume. Expressed in a flowchart:



- Determine which *looks block* you need for this.
- Run and test your code.

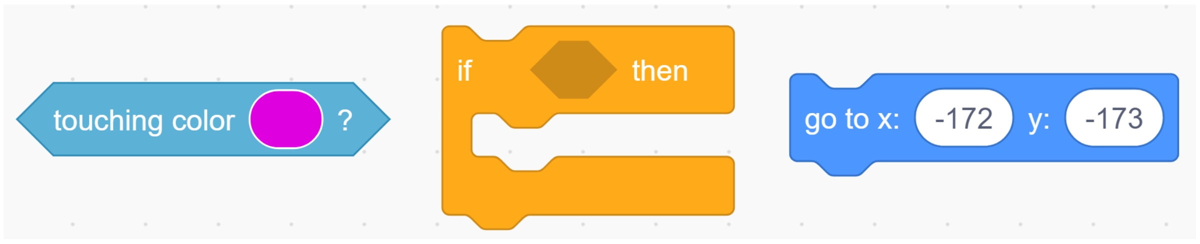
*

The general variant of a selection statement offers the possibility to also perform an action if the condition is false. In the previous example, nothing happened in a false condition. In a flowchart an if-then-else statement is indicated as follows.



If the condition is true, the 'then branch' (labeled true) is executed. Otherwise, the 'else branch' (labeled false) is executed.

Scratch has the following blocks for moving the mouse and checking that the crab hits the walls of the maze.



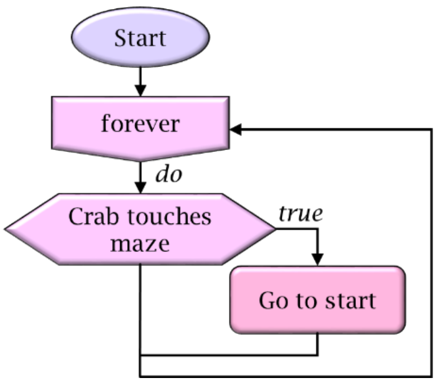
Pair task

Determine which blocks you need and what the order of blocks has to be.

- If the crab touches the sides of the maze, how to move the crab back to the beginning.
- Add the appropriate blocks to your code
- Run and test your code.
    - Does it work? If not, try to figure out what goes wrong and fix it.

*

We have already seen how we can express in a flowchart that a certain part of an algorithm must be repeated a number of times. It is possible that we want to indicate that an algorithm must always be repeated and therefore never stops. In our crab game we can use this to check again and again whether the crab touches the walls and if necessary should be placed back to the beginning. In a flowchart:



Note that this flowchart has no end node.

Pair task

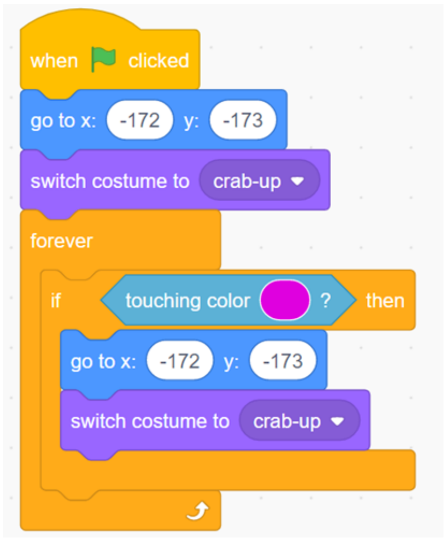
- Use the forever loop (from the *control blocks*) to handle the crab's movement.
- Find out what needs to be done if the crab hits one of the walls.
- Run and test your code.

*

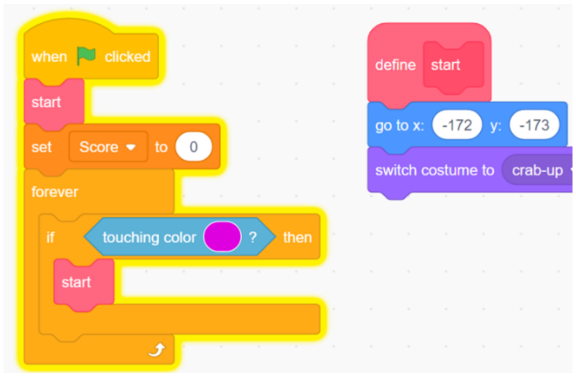
A natural way to solve large problems is to break them down into a series of sub-problems, which can be solved more-or-less independently and then combined to arrive at a complete solution. During the design stage of an algorithm, as a problem is subdivided into sub-tasks, the problem solver should have to consider only what a sub-algorithm is supposed to do globally and not be concerned about the details of that sub-task. This separation of concerns is known as abstraction. Through the process of abstraction, a programmer can hide all the details about sub-algorithms in order to reduce complexity and thus making the program more understandable.

Scratch uses abstractions to make programming easier by offering a multitude of basic building blocks whose complex underlying implementations remain hidden. Scratch programmers can introduce abstractions by adding new blocks to the program themselves.

For example, consider the following program snippets, both of which may be a solution to the previous problem. On the left you see the solution where everything is worked out straightforward. On the right, abstraction is used, by introducing the moving of the crab and changing the costume as a new building block (with the name Start) and then using it in the main program.



```
when clicked
go to x: -172 y: -173
switch costume to crab-up
forever
    if touching color ? then
        go to x: -172 y: -173
        switch costume to crab-up
```



```
when clicked
start
set Score to 0
forever
    if touching color ? then
        start
```

```
define start
go to x: -172 y: -173
switch costume to crab-up
```

Pair task

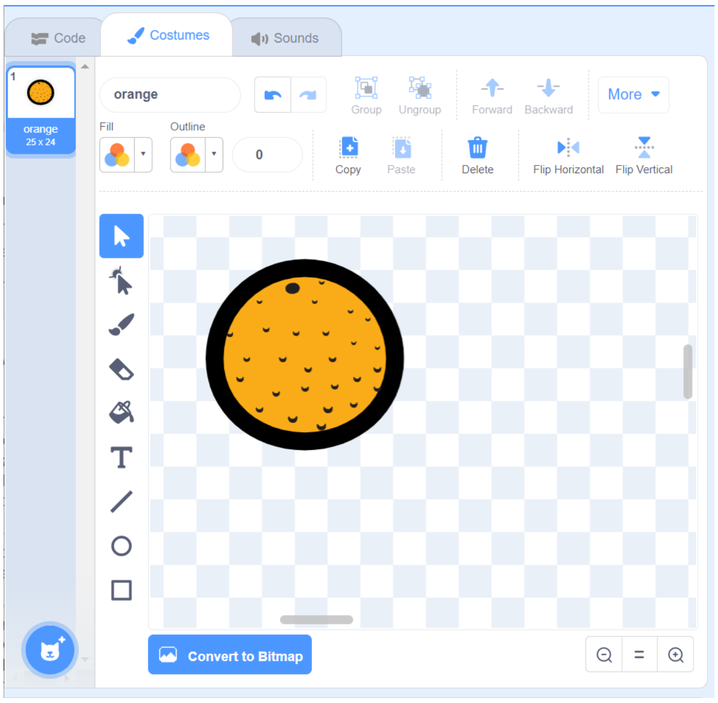
Go through your Scratch scenario and apply abstraction by replacing (detailed) code fragments with self-defined blocks

*

A player of the game should now be able to control the crab with the arrow keys and the crab will respond correctly when a wall of the maze is hit. We are now going to make the game more attractive by adding challenges. At first, we limit ourselves to oranges that we spread over the maze and that are eaten by the crab. This happens as soon as the crab touches them. Eating an orange provides a bonus point. The object of the game is now to earn as many bonus points as possible.

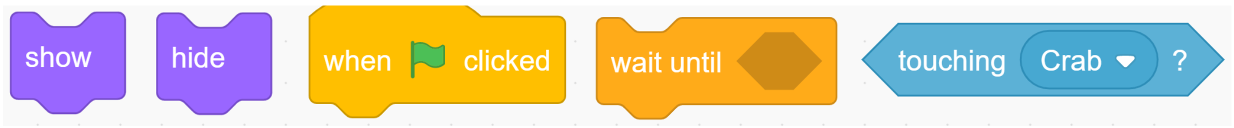
Pair task

- Select the orange sprite and scale the orange so that the size is in proportion to the crab.



We now run into an interesting problem: who do we make responsible for detecting and handling the interaction between crab and orange? In the real world, this will always be the crab, but here we have a choice. We can give the orange more responsibilities than in reality. And we will now use this interesting option: we will make the orange detect whether it has been found by the crab and let itself be eaten. We achieve the latter by hiding the orange.

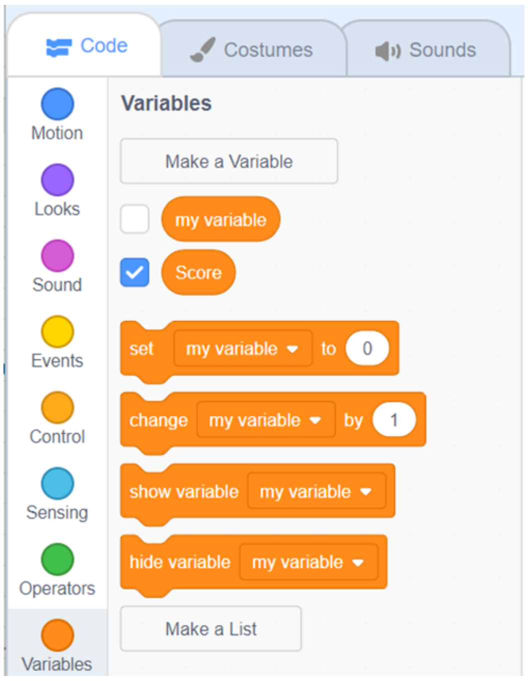
- Below you see the blocks that you will (probably) need to program the oranges. Use these blocks to get the desired behavior.



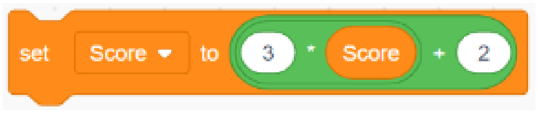
- Run and Test your code!
  - Is there anything else that should happen at this point?

*

We also want to add some way to the program to keep track of the score. We will use a so-called variable for this purpose. In programming languages, a variable is a container that can hold one piece of information at a time, like a word or a number. Being able to hold this bit of information allows us to reference and manipulate it at different places in a program. This ability makes variables incredibly useful.



How do we make a variable in Scratch? Before we can use a variable we must first create it using the "Make a Variable" button in the Block Palette. The value that a variable in Scratch can contain is either a text or a number. Furthermore, there are blocks to give an (initial) value to a variable and to change the value during the execution of the program. On the right you can see which blocks you can use for manipulating variables. By the way, the value field in, for example, the set block does not necessarily have to be a number: more complex expressions composed with operators from the operator palette are also allowed here. E.g. suppose Score has the value 4. After the execution of



Score will have the value 3 * 4 + 2 = 14.

Pair task

- Add a variable with name Score to your program.
- Think about what value this variable has at the start and consider how you can set this value.
- Also consider where and how the value of Score should be adjusted and which block can be used for this
- Run and Test your code!
- Duplicate the orange (in the sprites window) a number of times (e.g. 6) and place these copies somewhere in the maze.

UNIT 4: Teaching and Learning Computational Thinking

Total time: 8 hours

Teaching and learning about computational thinking can take place in essentially two styles: without computers ('unplugged') and with computers ('plugged'). In this unit you will experience both ways and you will learn about design principles for instructional strategies for computational thinking.

Contribution to the learning outcomes

Learning outcomes
• describe characteristics of unplugged and plugged teaching and learning activities for computational thinking • describe essential elements of instructional strategies for computational thinking • recognize such elements in concrete teaching materials and activities

Activity 4.1 Bebras Tasks

45

Bebras

Bebras is an online contest and challenge, organized by an international community. The elements of the contests are so-called Bebras tasks, each covering one or more computational concepts. In this activity you will explore Bebras tasks both as units of the contest and as 'unplugged' learning activities for computational thinking.

Bebras
International Challenge on Informatics and Computational Thinking

Pair Task

- Go to the local Bebras website and select up to five example Bebras tasks (try to include some variation).
- Carry out the tasks individually and then compare and discuss them with your partner.

- Can you classify the tasks in terms of the computational thinking steps and activities you learned in this module?

Bebras Tasks as learning activities

Bebras tasks are categorized in terms of the computational concepts being covered, each task carries an explanation of the connection between the task and computer science.

Valentina Dagienė (the 'mother of Bebras') and Sue Sentance investigated the use of Bebras tasks as CT learning activities:

Dagienė, V., & Sentance, S. (2016). It's Computational Thinking! Bebras tasks in the curriculum. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 28–39).

Pair Task

• Read the article.

• Review the concepts and activities you encountered in units 2 and 3. For each of these modules, find an example of a computational concept and a matching Bebras task.

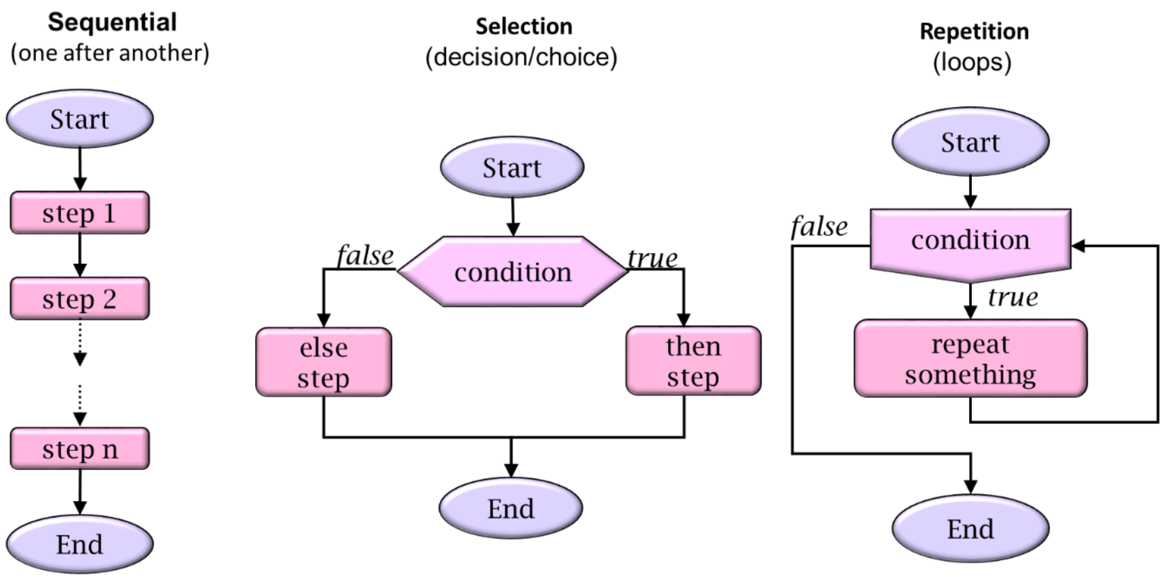
Activity 4.2 Pathfinding: paths in a maze

Maze solving is the process of finding a path through the maze from the start to finish. Some maze solving methods are designed to be used inside the maze by a traveller with no prior knowledge of the maze, whereas others are designed to be used by a person (or computer program) that can see the whole maze at once. Maze solving is a variant of a more practical class of problems that is known under the name of pathfinding. Given a number of locations that may be interconnected (for example, a collection of cities that are connected by roads), a pathfinding algorithm searches for a route that connects these locations with the intention of determining the best route (e.g. the shortest or cheapest route).

Our intention is to use these pathfinding problems to gain first insight into problem analysis and algorithm design without detailed knowledge of computer programming and programming languages. However, in order to be able to formulate possible solutions with sufficient precision, we will first use flow charts again.

Flowchart elements

Below you see the three basic building blocks with which flowchart can be composed.



Each flow chart is built up using the following three basic elements

1. **Sequence**: an ordered series of instructions that are executed one after the other.
2. **Selection**: it is determined on the basis of a conditional expression whether the then-branch (the branch with the label true) or the else-branch (the branch with the label false) should be executed.
3. **Repetition**: the body of the element (the branch with label true) is repeated as long as the condition evaluates to true. Then the program will continue by following the branch labeled false.

Travelling salesman problem

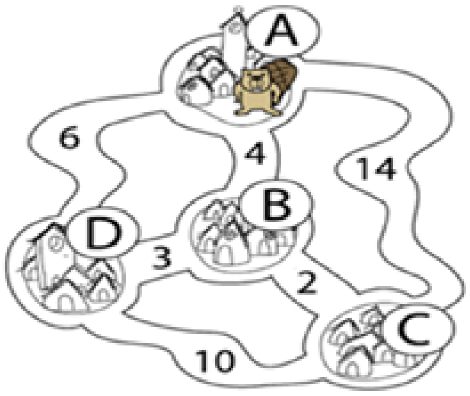
The so-called traveling salesman problem is a classic example of a task in which an optimal solution is sought.

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"

We are now going to look at a solution to this problem that, while not always finding the best route, is intuitive and easy to understand, namely the *nearest neighbor algorithm*. The nearest neighbour algorithm (NNA) lets the salesman choose the nearest unvisited city as his next move.

Pair Task

Beavers swim from city A, they go up visit at city B, C and D and then come back in A. They want to find the shortest route.



1. Which route do they find if they would use NNA?
2. Does NNA always give the right answer (i.e. for any list of cities and connections between those)? If not, give a concrete counter-example

*

Although the NA is intuitively clear, it can be difficult to formulate this algorithm precisely. We try to express this algorithm in a flowchart where we first have to determine which primitive instructions we can use for this. These primitives must be powerful enough on the one hand, but also sufficiently abstract on the other, so that we do not get caught in all kinds of details.

Pair Task

Specify the NNA in a flowchart.

Hint: Think first about the basic instructions/primitives that you are allowed to use.

60

Finding a path in a maze

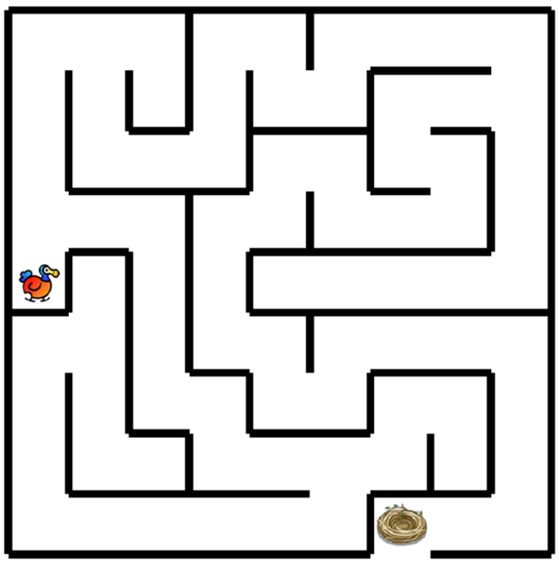
Finding a path in a maze is another example of a task that is easy to understand, but also challenging enough to solve. There are several variants of this problem: find a path out of a maze, find a path through a maze or find a path to a specific location inside a maze. First, we can note that these variants can be generalized to: find a path from position A to position B.

It is also important what we know about the maze while we search for the goal position B. In this assignment, we assume that we know nothing about the size and structure of the maze. We can walk through the corridors of the maze, looking one step ahead at a time. We therefore assume that at any point in the maze we can check whether we can walk straight (and therefore not stand in front of a wall) and possibly adjust our direction of movement.

The best-known rule for traversing a maze is the wall follower (also known as either the left-hand rule or the right-hand rule). If all walls of a maze are connected together then by keeping one hand in contact with one wall of the maze the traveler is guaranteed not to get lost and will reach the exit.

To make the problem more concrete, we use the following scenario where the maze traveler is represented by a dodo who is somewhere in the maze trying to find his nest hidden elsewhere.

Before we can specify a search algorithm, we must again indicate with which primitive instructions we can control the dodo.



We distinguish between *commands* (to have the dodo take a step forward or change the direction of movement) and *queries* (with which we can have the dodo provide local information about its surroundings).

- Commands:
    - *turn left*: turn 90 degrees counterclockwise
    - *turn right*: turn 90 degrees clockwise
    - *move*: move one square ahead
    - *lay egg*: lay an egg (at the current location)
- Queries/tests:
    - *can move?* can you take a step forward?
    - *nest found?* have you found the nest?

Pair Task

Specify the follow-the-left-wall strategy in a flowchart.

Pathfinding unplugged

Preparation

For example, using painter's tape, stake out a maze on the floor. Make sure the corridors are wide enough for a person to walk through easily. The maze may be a bit simpler than the maze in the picture. Exchange the flowcharts from the previous assignment. Designate a place in the maze as the final destination and one of you will be somewhere in the maze far enough away from this final destination.

Pair Task

One of you will be standing somewhere in the maze far enough away from the final destination.

- The other person reads the instructions from the flowchart and the one in the maze follows these instructions meticulously.
- Is the final destination reached? If so, also try out whether this applies to other starting locations in the maze. If not, what is wrong with the algorithm? Improve the algorithm and check whether the improvement has the desired effect.
- Return the improved version of the flowchart to their owners.

Learning Resources

- Scratch file: PathThroughMaze.sb3

45 **Pathfinding in Scratch**

Finally, you will implement the algorithm from the previous assignment in Scratch. We have created an initial scenario that you can use for this. Open the start scenario PathThroughMaze.sb3.

A brief explanation now follows. The primitive dodo instructions have been added to the scenario as separate blocks. The *layEgg* instruction is missing because it is not needed for this task. All other commands work as previously indicated. The queries have been realized in a special way, also because self-defined blocks in Scratch cannot yield any results. The test to check whether the dodo has found the nest can easily be realized with a predefined block from the sensing palette. We introduced a new block for the *canMove* test. To return a result, we use a variable that we, like the block itself, called *canMove* as well. So after you execute the *canMove* block you can inspect the value of the corresponding variable to see if the dodo can take a step or not. The variable *canMove* contains either the value 0 or 1. The value 0 indicates that no step can be taken while value 1 indicates that it is possible.

Pair Task

The starter scenario contains a block called *tryToFindExit*. This block still does little: it just puts the message that the exit has been found on the screen.

- Implement this block by converting the flowchart into Scratch code.
- Run and test your program for other starting positions of the dodo as well. Adjust the code if necessary.

Activity 4.3 Instructional Strategies

In this activity you will explore strategies and pedagogical principles for computational thinking.

Looking Back

Pair Task

Discuss your experiences in the Pathfinding activity:

- What are the essential differences between the 'unplugged' and the 'plugged' variant?
- What challenges for your future students do you expect for each of these?

Instructional principles for CT

In a review article, Lye and Koh (2014) analyze 27 empirical studies on Computational Thinking, in particular studies on classroom activities involving programming. Among other things, Lye and Koh identify instructional approaches fostering computational thinking.

Lye and Koh grouped the approaches into four categories, which often appear in combinations: *reinforcement of computational concepts*, *reflection*, *information processing*, and *constructing programs with scaffold.*

Pair Task

1. Read the summary on the four categories of the document ''Instructional strategies for computational thinking'' (which can be found in the learning resources).
2. Analyze the computational thinking activities you carried out yourself in this module. Which of the strategies do you recognize? Discuss.

Pair Task

1. Read the summary on assessment in computational thinking (see the document ''Assessment in CT'').
2. Discuss which would be suitable assessment approaches for the activities you carried out in Unit 2 or Unit 3. Select one possible approach for each activity. How would you use these approaches to evaluate students' computational thinking skills?

Conclusion

Discussion Task

Compare your findings in class. Which aspects did you find easy, which ones were difficult? What did you learn?

Learning Resources

- A. Instructional strategies for computational thinking (Summary of instructional strategies for CT teaching and learning, based on Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51–61.)
- B. Assessment in CT

Overview of Learning Resources of Module O2

- Activity 1.1. - A. Brief introduction to CT
- Activity 1.1. - B. Definitions of CT
- Activity 2.1 - A. Gemeinschaft to Gesellschaft.pdf
- Activity 2.1 - B. The changing psychology of culture from 1800 through 2000
- Activity 2.1 - C. The changing psychology of culture in German‑speaking countries
- Activity 2.2 - zikavirusmodel.nlogo
- Activity 2.4 - Earth colour workbook.xlsx
- Activity 3.2 - crab-in-maze-start.sb3
- Activity 4.1 - Dagiene and Sentance 2016 (Dagienė, V., & Sentance, S. (2016). It's Computational Thinking! Bebras tasks in the curriculum. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 28–39)).
- Activity 4.2. PathThroughMaze.sb3
- Activity 4.3 - A. Instructional strategies for computational thinking (Summary of instructional strategies for CT teaching and learning, based on Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51–61.)
- Activity 4.3 - B. Assessment in CT