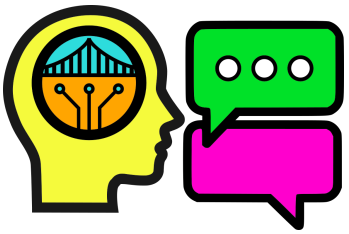


# Code with TurtleArt

## Lesson summary

**Target Audience:** *10-16 years old. Beginners.*
**Duration:** *90 minutes.*
**Learning goals:**
Students will take part in an engaging hands-on activity where they will apply their programming skills to draw geometric figures made with code blocks. By using samples and modifying them, the students learn to create their own figures. The goal of this activity is to advance their understanding of basic programming and making a graphical output.
The objective is to help students develop computational thinking skills by:

- Develop programming skills to solve problems (algorithms and coding).
- Decomposing the problem into manageable parts (problem decomposition).
- Design and apply algorithms to process data (algorithmic design and process automation).

**Online or offline:** online
**Computational Thinking:**

- **CT-foundations**:
    - Decomposition: Breaking down clues and steps to isolate password components
    - Abstraction: Ignoring irrelevant clues or distractions
    - Algorithmic Thinking: Systematically testing possibilities to uncover the correct password
    - Logical Reasoning: Deducting correct choices based on logical elimination

# Materials

- Computer.
- Internet.
- Link: [playfulinvention.com/webturtleart](playfulinvention.com/webturtleart)

# Preparation

1. **Divide participants into small groups of 2 or 3 people.** If the groups are larger, it may be difficult for everyone to interact.
2. **Provide each group with a computer**

# Lesson Description - Code with TurtleArt

**Introduction to Computational Thinking** (10 minutes)

Ask the students:
- What do you know about how computers and telephones work?
- Can computers/telephones think for themselves? (Why or why not?)
- Who controls what a computer does?
- Are computers creative?
- Can computers solve problems?

Explain to the students that they are going to work on Computational Thinking. Simply put, this involves learning how to get a computer to solve a problem for you. It is not merely programming, but also, for example, learning how to break down a problem into pieces, or recognizing patterns so you can better solve a problem.

There are four main foundations of CT:

- Decomposition➔ dividing a problem into small pieces
- Pattern recognition ➔ looking for similarities or patterns within those small pieces that can help you solve the problem.
- Abstraction ➔ distinguishing between the main and secondary issues. What is really important to solve the problem?
- Algorithms ➔ coming up with step-by-step instructions to solve the problem.

In this lesson you will be introduced to pattern recognition

**Main Activity – Coding with TurtleArt**

Round 1 (20 minutes)

- Short introduction to the interface of TurtleArt (See Appendix 1).
- Copy the code from the Sample 1 (See Appendix 2).
- Run the programme.
- What happens if you change the numbers in the pink blocks?
- Can you make a simple cross instead?
- Can you change the colour?
- Explain how it works for a fellow student.

Round 2 (10 minutes)

- Copy the code from the Sample 2 (See Appendix 2)
- Run the programme
- What happens if you change the numbers in the pink blocks?
- Explain how it works for a fellow student.

Round 3 (10 minutes)

- Copy the code from the Sample 3 (See Appendix 2).
- Run the programme.
- What happens if you change the numbers in the pink blocks?
- Explain how it works for a fellow student.

Round 4 (10 minutes)

- Can you make a square?
- Can you make a triangle?
- Can you make a square spiral (advanced)?
- Explain how it works for a fellow student.

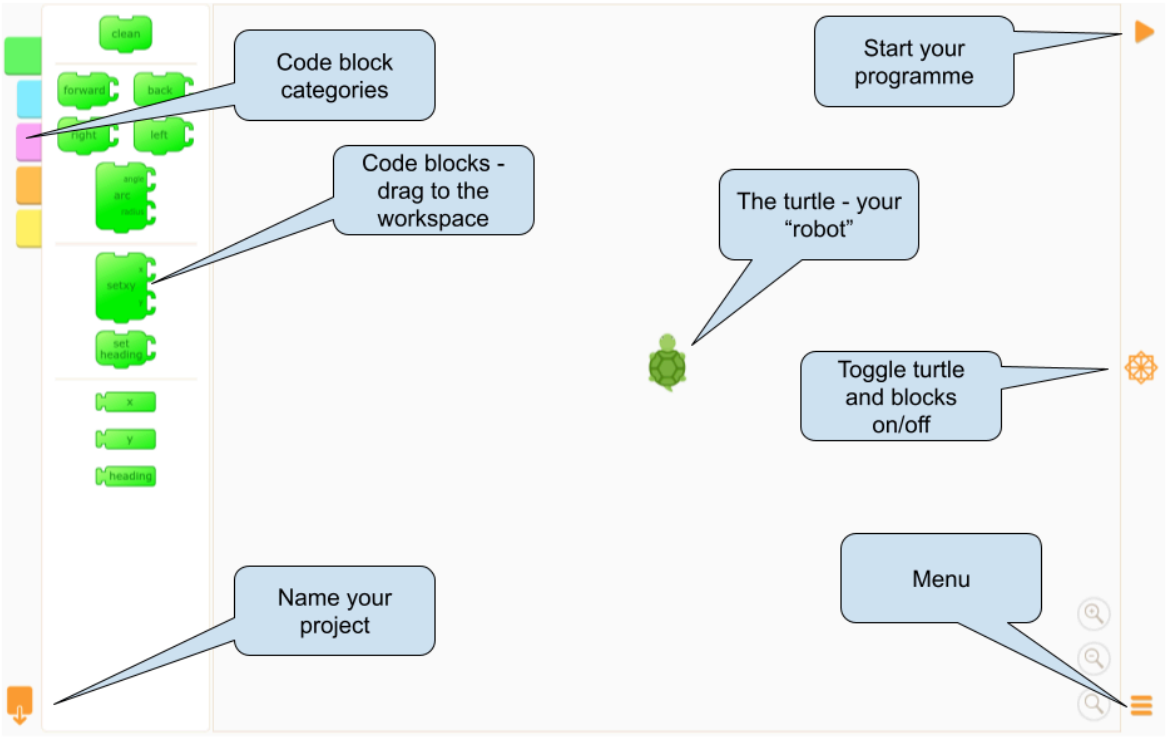
Round 5 (10 minutes)

- Make your own work of art.
- Explain how it works for a fellow student.

**Debrief and Evaluation (5 minutes)**

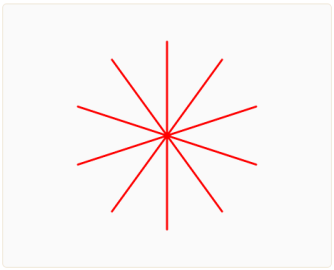
- What did we learn?
- Was it difficult to understand how the code worked?
- Was it difficult to create your own art?
- Where can we use what we learned?
- Etc.

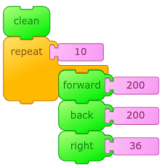
# Appendix 1 - Interface explained



# Appendix 2 - Basic samples

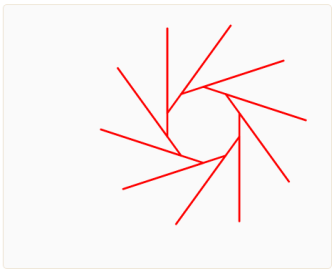
## Sample 1

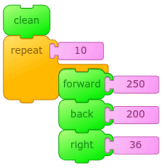





**forward 200** draws a line. Then **back 200** retraces that line and brings you back to the starting point.
Repeat the line 10 times with **right 36** in between. This makes a star with the lines evenly spaced. The total amount of turning is 10*36=360 degrees, the number of degrees in a circle.

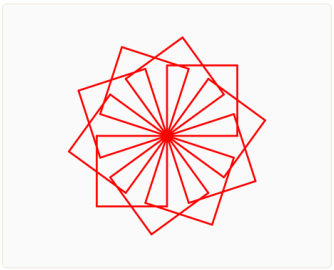
## Sample 2

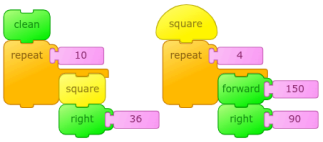




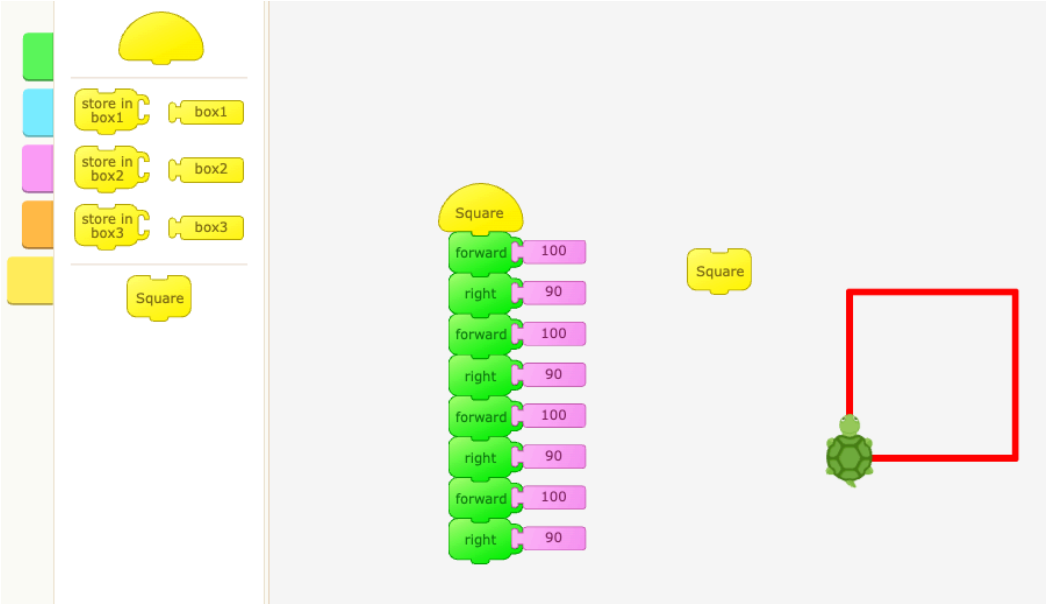

**forward 250** draws a line. **back 200** retraces most of it. The Turtle doesn't quite get back to its starting point. Each line starts at a different position than the previous one.

## Sample 3





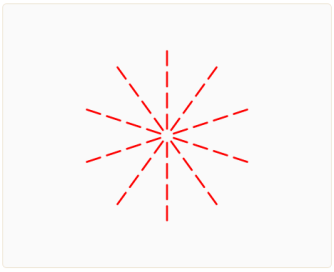

You can name stacks. Pull out an empty hat block (the round yellow one). Click on it and type in a name. A new block to use that stack will appear in the blocks palette. In this sample, the **square** block isn't there when you start. It only appears after you have pulled out a hat and named it **square**.



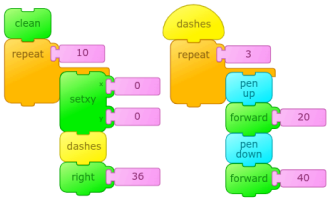

You can type directly into the text block connected to the yellow "Show" block (which may resemble a mushroom). This allows you to display your custom text on the screen when the program runs. Simply click on the text block and type your message

## Sample 4

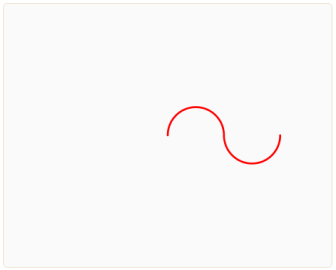
sqara

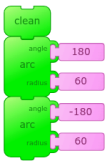


You can make dashed lines with **pen up** and **pen down**.

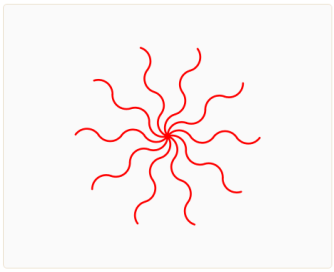


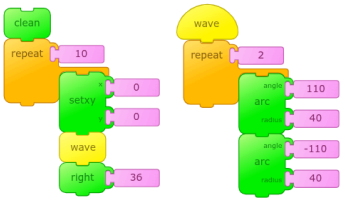
# Appendix 2 - Advanced samples



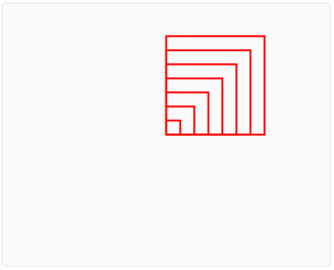


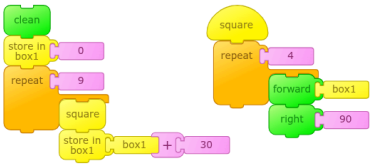

An arc is a part of a circle. There are two inputs to the **arc** block: radius and angle. The radius is the size of the circle. The angle is how much of the circle to draw. An angle of 360 draws the whole circle.



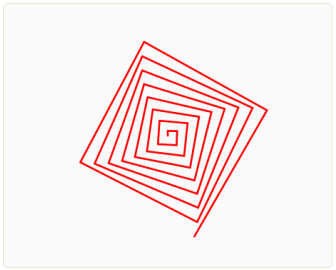


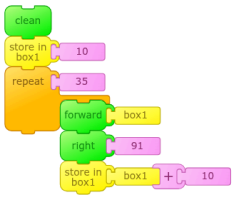

Once you have an interesting element you can "spin" the element with a **repeat** and a **right**. In this case, the element is a double wave with each of the waves made with two **arc** blocks.



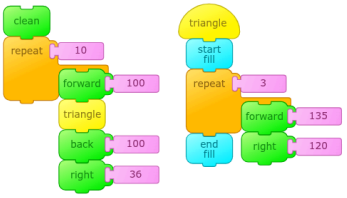



**store in box1** lets you save a number. **box1** recalls that number. You can use the boxes to make images that have repetiton with variation.



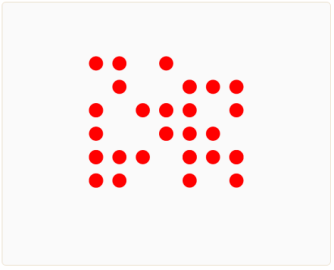


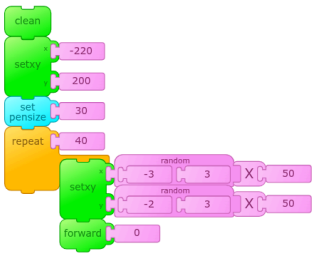

**box1** can be used to create a succession of lines of increasing length. Put an angle between these lines and you get a square spiral.




You can fill areas with **start fill** and **end fill**. The blocks between **start fill** and **end fill** define the area to be filled.

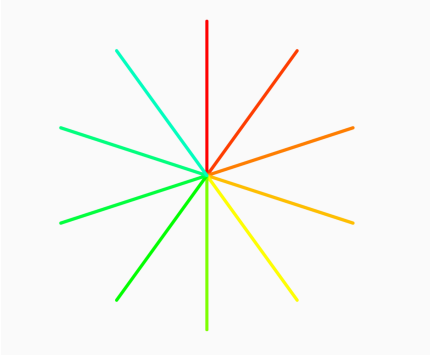
You can fill areas with **start fill** and **end fill**. The blocks between **start fill** and **end fill** define the area to be filled.

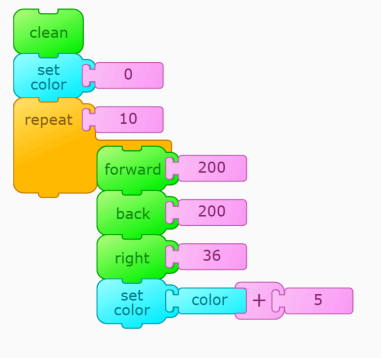


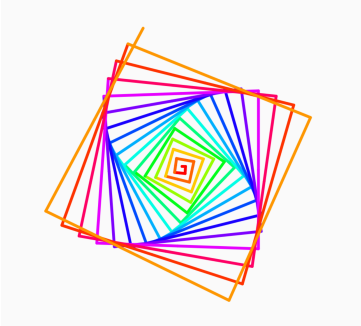


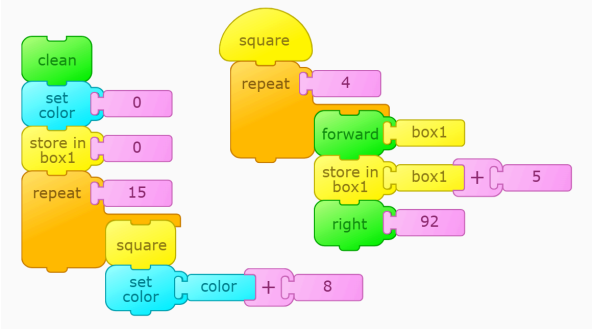

Each time the program runs, it uses a **random number** to decide where some of the dots should be placed. Because of this, the picture will **look a little different every time**, even though the code is the same.

This is important to know, because students might think they made a mistake if their result doesn't match the example exactly. But the variation is **intentional**—it's part of how randomness works in programming.



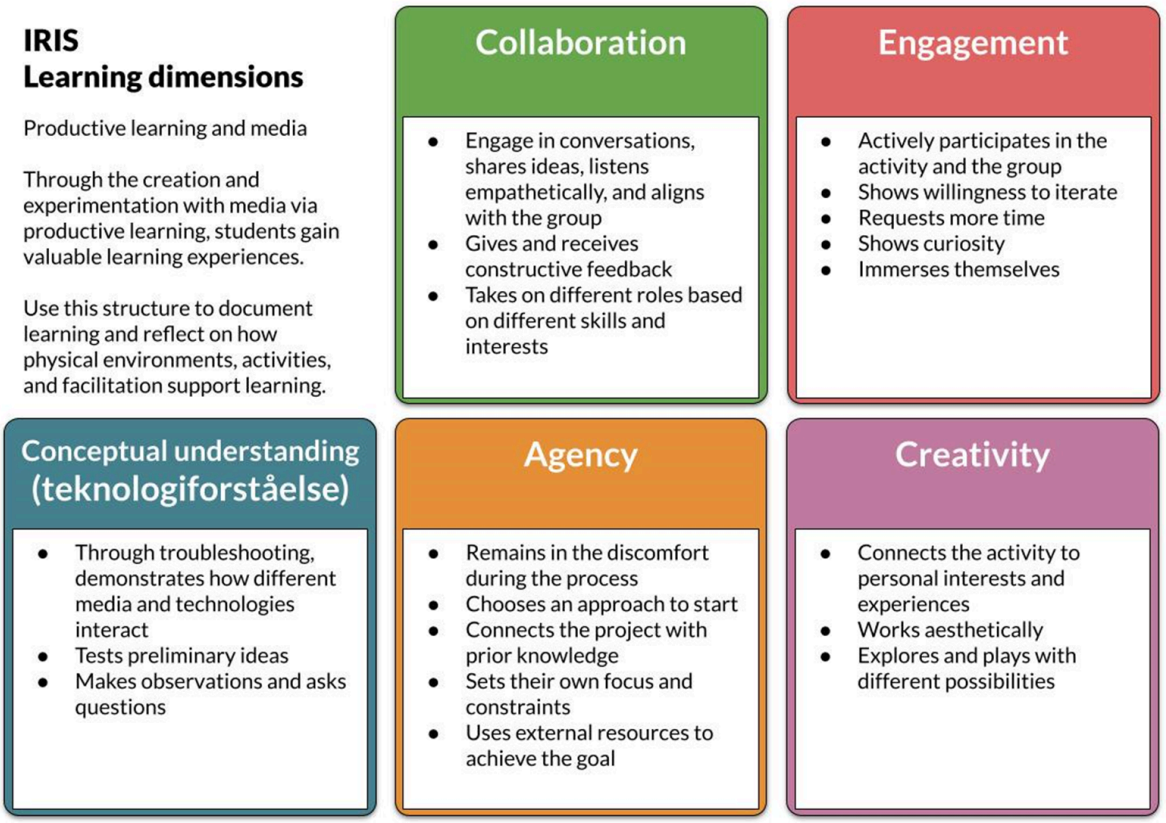




# Appendix 4 - Learning Dimensions

Learning Dimensions are developed at The Thinkering Studio at Exploratorium in San Fransisco, US and our work toward reflections are deeply inspired by their work.




**What Do We Want Participants to Gain from Our Activities – and How Do We Ensure That It Happens?**

Quality in teaching is both a complex and sometimes controversial topic. Therefore, we use the **Learning Dimensions** – a dynamic tool developed by the Exploratorium in San Francisco. We use the Learning Dimensions to document learning and to evaluate whether we succeed in providing participants with a valuable and meaningful experience through our activities.

**Our Starting Point**

When we develop and evaluate an activity, we always ask ourselves:
**What do we want to learn or find out through this activity?**

To ensure a thorough and nuanced process, we work in three phases:

- **Pre-Meeting**
  Before the activity, we hold a meeting to determine which learning dimensions and which (two) indicators we want to focus on.

- **Green, Yellow, and Red**

  During the activity, an observer is present to take notes based on the agreed dimensions and indicators.
  Immediately after the activity, we gather for a short reflection session, assessing the activity using a red, yellow, and green model.
  What worked well, what worked less well, and what didn't work at all?

- **Post-Meeting**

  The week after, we meet with the person or people who observed the activity for a deeper analysis of how the learning dimensions were reflected in the activity.

**A Flexible Assessment Framework**

We assess our activities using a range of learning dimensions and associated indicators. The areas of focus depend on the individual activity and therefore vary from time to time. However, we also compare the learning dimensions across different activities to identify patterns—whether certain dimensions are being neglected, need updating, or could be improved.

The learning dimensions are crucial for assessing the participants' outcomes from the workshop, but also for gaining insights into the breadth, depth, and level of our activities in general.

**A Tool for Continuous Improvement**

The goal of working with the learning dimensions is not to reach a final answer, but to ensure ongoing development. Our approach is to **"be on the right path"**, rather than to meet a fixed, predefined goal. The tool is therefore **not** a measuring device, but should remain a dynamic aid that supports educators in creating inspiring and engaging learning experiences.