

DEPARTMENT OF COMPUTING AND SOFTWARE
McMASTER UNIVERSITY

Recursion Creator

Jiawen Wang

Supervised by

Dr. Christopher Anand

Associate Professor, Department of Computing and Software
McMaster University

May 3, 2023

Acknowledgement

Foremost, I would like to express my sincere gratitude and appreciation to my supervisor, Dr. Anand, for his continuous support and patience. His immense knowledge and guidance have helped me significantly in all my research during my study. I could not have imagined having a better supervisor.

In addition, I would like to give my greatest appreciation to Sheida Emdadi, who helped to proofread my report and improved my report significantly with her fantastic knowledge of English.

My sincere thanks also go to my fellow friends Vaitheeka, Varun and Akshay for all their assistance on all the projects we have worked on together. Besides, I would like to thank Ms. Anand for hosting the Summer Camp and all the students participating in the summer camp.

Last but not least, I would like to express my deepest gratitude to my caring, loving and supportive mother; without her support and encouragement, I would never have had the courage to pursue my degree overseas.

Contents

1	Literature Review	1
1.1	Challenges in learning Recursion	1
1.1.1	Different approaches to learning Recursion	1
1.1.2	Learning with previous programming experiences	2
1.1.3	Learning with incorrect preconceptions	2
1.1.4	Learning with algorithmic approach	3
1.1.5	Misconception about the base case	3
1.2	Assessment dimensions	4
2	Recursion Creator Tool	5
2.1	Overview of the tool	5
2.1.1	User Interface	5
2.1.2	Recursion Shapes	5
2.2	Code Base	6
2.2.1	Core Types	7
2.2.2	Modules	13
2.2.3	Code Architecture	16
3	Recursion Creator Camp	18
3.1	Curriculum	18
3.1.1	Review of Elm programming	18
3.1.2	Introduce Recursion	19
3.1.3	Recursion shape challenge	19
3.1.4	Assessment	20
4	Conclusion	21
5	Appendix - Summer Camp Slides	22
	References	32

1 Literature Review

Recursive programming was first introduced with ALGOL 60 in 1960 [Daylight 2011]. Since then, Recursion has been considered one of the most challenging topics to learn and teach. Researchers and educators suggest that Recursion is “one of the most universally difficult concepts to teach [Gal-Ezer and Harel 1998]”; meanwhile, students consider Recursion to be “obscure, difficult, and mystical” as indicated in the book [Roberts and Roberts 1986].

In this section, we will discuss some of the most typical challenges found in learning Recursion and the assessments we can use when teaching Recursion.

1.1 Challenges in learning Recursion

In the paper [Roberts and Roberts 1986], the authors provided a thorough literature review on the previous research regarding teaching students Recursion. From the collected research, they found the most common challenges to be

- Different approaches to learning recursion
- Learning with pre-programming experiences
- Learning with incorrect preconception
- Learning with an algorithmic approach
- Student’s misconception about the base case

1.1.1 Different approaches to learning Recursion

In a study conducted by [Kessler and Anderson 1986], the authors suggest that the lack of real-life examples of Recursion forced students to learn Recursion from code examples; however, the human brain does not usually think recursively, and therefore, the lack of examples from the physical world would significantly hinder students’ ability to grasp the concept of Recursion intuitively.

In addition to the lack of intuitive examples, Recursion can also take many forms and appear anywhere within a code block, making it nearly impossible for students to recognize Recursion by classifying them into specific patterns.

When students start to learn to program, one of the promising approaches is through debugging and tracing. For a method/code block that uses Recursion, debugging and tracing would require Recursion itself, adding more confusion for students to understand Recursion.

1.1.2 Learning with previous programming experiences

In [Lee and Lehrer 1988], Lee and Lehrer conducted two experiments on teaching Recursion to students with and without programming experiences. The study concludes that previous programming experiences may negatively impact students' ability to learn Recursion. The study asked two groups of students - students with programming experiences in BASIC and students without any programming experiences to complete three programming assignments requiring the use of Recursion after both groups had received 1.5 hours of instruction each week for eight weeks. Lee and Lehrer were able to identify some common errors in both groups, such as:

- Unnecessary repetition of statements
- Inappropriate use of conditionals
- inefficient use of logical operations

However, a few errors only appeared within the group with programming experiences, such as using “goto” for Recursion and unnecessary nesting. The result suggests that prior programming experience will not necessarily improve students' understanding of Recursion. Oppositely, it might have some negative knowledge transfer regarding learning Recursion.

1.1.3 Learning with incorrect preconceptions

In the research [Booth 1992], 14 students were asked to teach the author about Recursion by solving a textbook problem to search a list. And then, those 14 students were examined with a searching problem - returning the position of a specified element. The author categorized the solutions into below categories:

- Solutions using a counter variable;
- Solutions using support functions to count;
- Solutions using direct Recursion

Only a few students could solve the problem using Recursion; most students devised solutions using their preconceived notions or experience. Booth concluded that “Learners use problems to challenge their preconceptions, eventually leading to mastery as well as using problems to reinforce incorrect conceptions, which in turn led to failure until a new teaching or learning intervention was made.[Booth 1992]”.

Learning Recursion solely through hands-on practice without initially being introduced to the proper fundamental Recursion concepts may compromise the effectiveness of students' learning experience. Since students are already preoccupied with false preconceptions, solving problems only reinforces those incorrect conceptions.

1.1.4 Learning with algorithmic approach

Most students have confidence in understanding iteration. Researchers suggested it might be more straightforward and intuitive to teach Recursion by comparing it with iteration [McCauley et al. 2015].

In [A. C. Benander, B. A. Benander, and Pu 1996], Benander, Benander and Pu recruited 275 computer science major students as participants. The students were divided into two groups. Both groups were given two algorithms - The search algorithm and The Copy algorithm. The differences are that one group received the search algorithm implemented recursively and the copy algorithm implemented iteratively, while the other group received the search algorithm implemented iteratively and the copy algorithm implemented recursively. Students were asked to describe what each algorithm did in plain English, and the experiment was repeated for three years with the same group of students.

Benander, Benander and Pu found that initially, for the search algorithm, students were more likely to understand the recursive version better than the iterative version, whereas, for the copy algorithm, the trend was the opposite, with students demonstrating a greater understanding of the iterative version. Nevertheless, in years two and three, the results showed that, for both tasks, the recursive code comprehension rate was faster than that of the iterative code, with statistical significance [McCauley et al. 2015].

The study illustrates that an algorithmic approach to learning Recursion would benefit the students, eventually. However, it may have a learning curve for students to develop a deeper understanding of the concept gradually.

1.1.5 Misconception about the base case

The base case plays a crucial role in Recursion. Unfortunately, several studies have identified base cases as a source of misconceptions for students [McCauley et al. 2015].

In earlier studies, researchers found that students often struggled with the concept of the Base case. In [Close and Dicheva 1997], Close and Dicheva identified one misconception in LOGO recursive programming for students: students did not use any STOP command as the base case. [Segal 1994] also noticed that students often comprehend a base case as a stopping condition, neither returning values to the caller nor completing suspended routines. In [Dicheva and Close 1996], the result also revealed that students often assume the base case stopped all calls. In the later study, [Haberman and Averbuch 2002] focused on the role of base cases. The result indicates four common difficulties among students with Recursion.

- Ignorance of boundary cases such as empty lists or trees;
- Ignore out-of-range values;
- Not using base cases as terminating condition
- Redundant base cases

Understanding what base case is and how to use it in a recursive algorithm is one of the essential steps in thoroughly understanding Recursion. Still, the human brain is more used to comprehend iteration, which does not need the base case. This would increase the gap in understanding the base case and impede the comprehension of Recursion.

1.2 Assessment dimensions

One of the crucial steps in learning or teaching is assessment. Assessment allows students to identify their weaknesses and enables teachers to gauge the effectiveness of their instruction in conveying knowledge to their students.

In [McCauley et al. 2015], the study proposed that we can assess students' progress from the below aspects.

- **Comprehension**
This measures how well a student can describe the concept of Recursion and whether the student would use the recursive solutions in solving problems.
- **Evaluation**
Evaluate whether students have the ability to trace the flow of a recursive invocation and correctly predict the results.
- **Construction**
Evaluate whether students are able to create recursive solutions.

2 Recursion Creator Tool

Chapter 1 discussed the challenges in learning Recursion and how to measure students' progress in understanding Recursion. We have designed a Recursion Creator Tool using Elm to better assist students in intuitively understanding Recursion. This chapter will give an overview introduction to this tool and its design.

At any time, you can try the tool

<http://www.cas.mcmaster.ca/~anand/RecursionCreator.html>

2.1 Overview of the tool

Recursion Creator provides an interactive interface for students to create various shapes using Recursion; different challenges of drawing recursive shapes are displayed in the Recursion Creator. To use Recursion Creator, students do not necessarily require comprehensive programming experience. They can add or decrease the number of shapes and the size of each shape only by manipulating the button to change the variables in the pre-created code. There are, in total, 14 different shapes shown as challenges, and each challenge is more complicated than the previous one.

2.1.1 User Interface

The interface can be divided into four parts, as shown in Figure 1. The upper left part is the Canvas. The shape is drawn on the canvas, the “+” and “-” buttons help the user zoom in and out, and the victory gesture enables the shape challenge. After enabling the challenge mode, green dotted lines will appear in the canvas, as shown in Figure 2. To complete the challenge, students must draw the shapes that follow those dotted green lines. The left corner is the generated code. After the shapes have been drawn, the complete code will be generated, and the code will be copyable. Students can copy the code to any editor to make adjustments or directly paste it into an Elm IDE to run the code.

Students can change some variables in the code shown in the upper right region. Only variables with blue rectangles can be changed. After clicking on the variable with a blue rectangle, a transparent blue slider will appear near the variable, allowing students to change the value. The interface is illustrated in Figure 3 below. The “+” and “-” are also used for zooming in and out. The right corner is the menu for all the challenges. By clicking each icon, the shape will be changed into a new shape for the new challenge, and the complexity of the shape increases.

2.1.2 Recursion Shapes

As discussed in Chapter One, multiple studies claimed forcing students to learn Recursion directly from code would impede their understanding of Recursion, especially for students

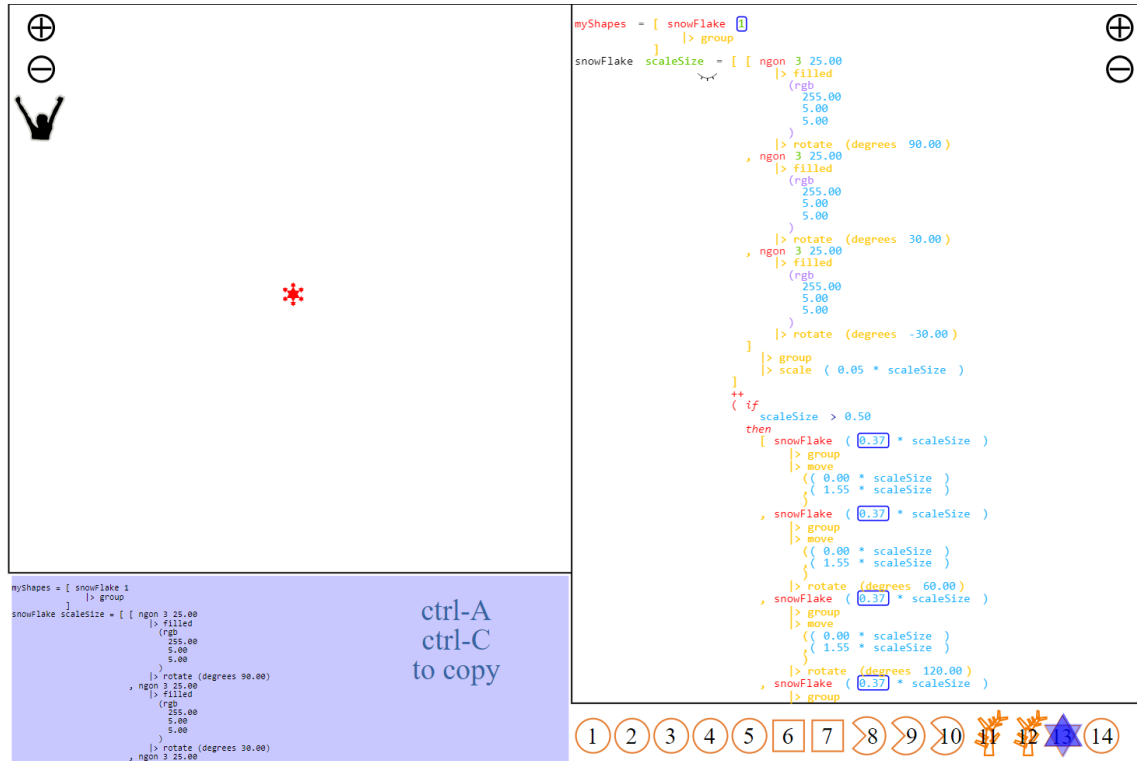


Figure 1: Recursion Creator Interface

with less programming experience.

Learning Recursion directly from code could bring some unexpected results, such as reinforcing the misconceptions students already have. To avoid forcing students to be directly exposed to Recursion through code, the Recursion Creator created various shapes using Recursion, allowing students to intuitively observe the pattern first, then jump into the code for more details.

All the shapes drawn by the Recursion Creator have a base shape. The base shape will always be presented to students for each challenge. The challenge could be as easy as five repetitive circles, as shown in Figure 4, which only requires students to determine how many circles need to be drawn using recursion. Once students get comfortable with the basic concept of Recursion, the shape challenge will combine multiple shapes, as shown in Figure 5. In this challenge, besides the numbers of rectangles and circles, students must consider the sizes, distance and rotation. The challenge can also be as complicated as drawing a coral using Recursion, as shown in Figure 6. Because every shape is drawn using Recursion and the recursion level can directly be seen through the shapes, it would be easier for students to observe the base, the recursive level.

2.2 Code Base

Recursion Creator is written using Elm. Elm is a functional programming language that can compile into JavaScript. Similar to Haskell, Elm is a typed language. Before we created Recursion Creator, most of our students had already been exposed to Elm to draw different shapes, write stories or write music programs. All the students are capable

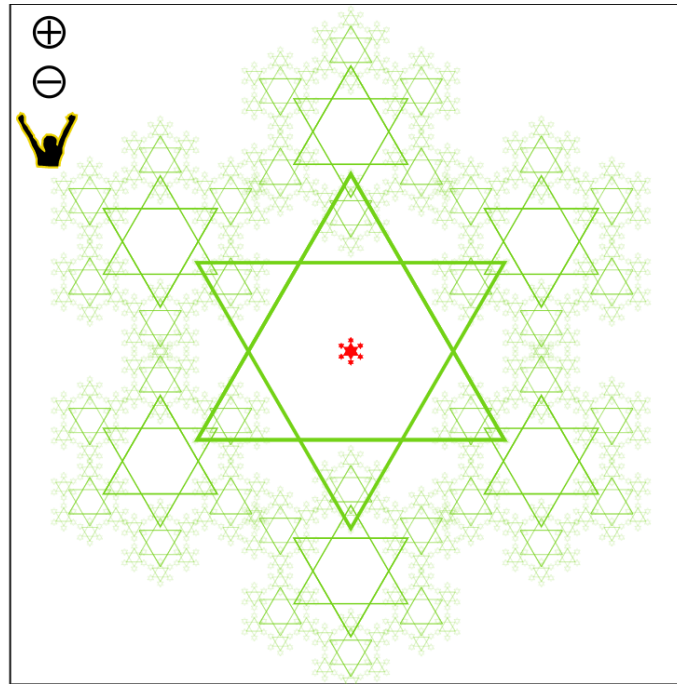


Figure 2: Challenge enabled

of using the library `GraphicSVG` and the IDE <https://macoutreach.rocks/> to program in Elm.

This section will briefly discuss the types we created for Recursion Creator. In addition, a detailed introduction will cover the modules we created. Finally, yet importantly, an overview of the architecture of the Recursion Creator code base will be provided.

2.2.1 Core Types

The most important types are `StackFrame`, `ElmStencil`, `ElmShape` and `ElmExpr`, shown below

```

1 type StackFrame
2   = SF (Dict Int ElmExpr) StackFrame
3     | TopLevel (Dict Int ElmExpr)
4
5 type ElmStencil
6   = Circle ElmFloat
7     | Rect ElmFloat ElmFloat
8     | Text ElmString
9     | Coral ElmFloat
10    | Polygon ElmFloat
11    | Ngon ElmInt ElmFloat
12
13 type ElmShape
14   = Filled ElmColor ElmStencil
15     | Outlined ElmLineTypes ElmFloat ElmColor ElmStencil
16     | Group (ElmList ElmShape)
17     | Move ( ElmFloat, ElmFloat ) ElmShape
18     | Rotate ElmFloat ElmShape
19     | Scale ElmFloat ElmShape

```

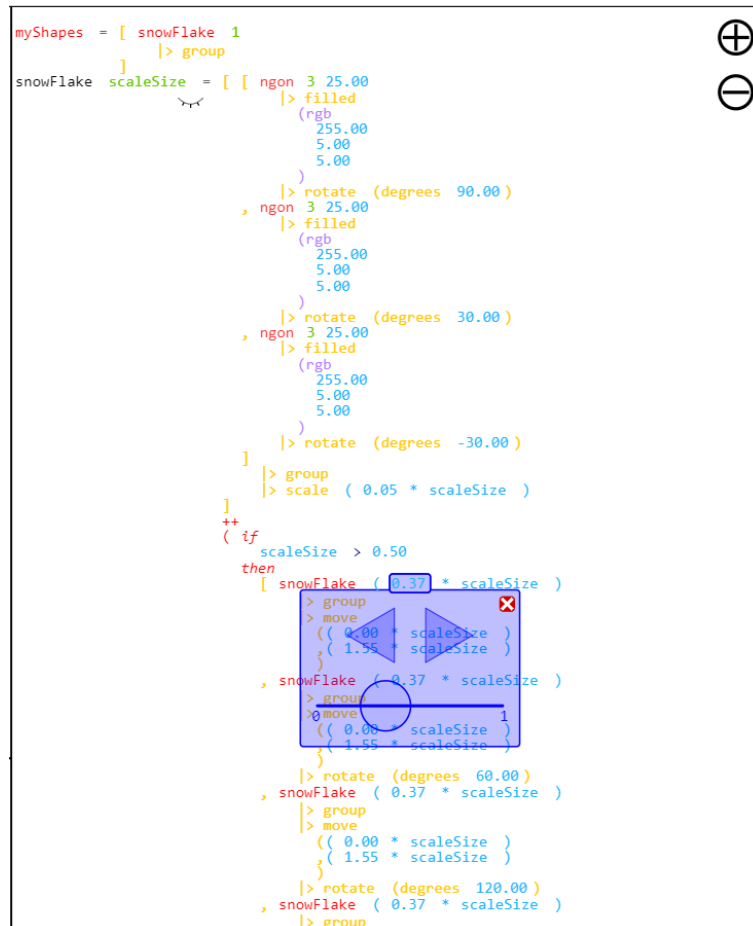


Figure 3: Code Panel with popup showing one Float value being adjusted



Figure 4: Recursive Circles

```

20 | CaseListString (ElmList ElmString) {- head::tail pattern -} ( ( VarName,
    BindIdx ), ( VarName, BindIdx ), ElmShape ) {- empty list pattern -}
    ElmShape
21
22 type ElmExpr
23   = ExprInt ElmInt
24   | ExprFloat ElmFloat
25   | ExprString ElmString
26   | ExprShape ElmShape
27   | ExprStencil ElmStencil
28   | ExprListShape (ElmList ElmShape)
29   | ExprListFloat (ElmList ElmFloat)
30   | ExprListInt (ElmList ElmInt)
31   | ExprListString (ElmList ElmString)
32   | ExprLambda ElmType ElmType { varName : VarName, eyeState : Int, bindIdx :
    BindIdx, elmExpr : ElmExpr } --( VarName, BindIdx, ElmExpr )
33   | ReducedInt Int
34   | ReducedFloat Float
35   | ReducedString String

```

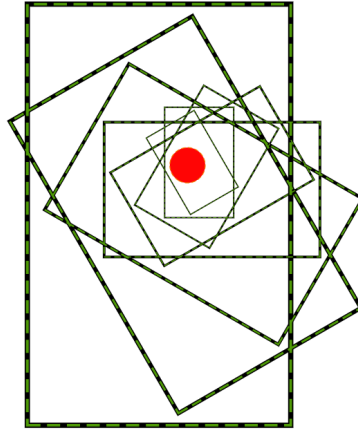


Figure 5: Rectangle combined with circles



Figure 6: Recursive Coral

```

36 | ReducedShape (Shape ElmViewMsg)
37 | ReducedStencil Stencil
38 | ReducedListShape (List (Shape ElmViewMsg))
39 | ReducedListFloat (List Float)
40 | ReducedListInt (List Int)
41 | ReducedListString (List String)

```

- StackFrame

`StackFrame` is the basic structure for drawing the shapes and the code we can see on the right side of Recursion Creator. It can have two values `TopLevel (Dict Int ElmExpr)` and `SF (Dict Int ElmExpr) StackFrame`. `TopLevel (Dict Int ElmExpr)` takes the payload of a dictionary mapping an integer to `ElmExpr`. `ElmExpr` is the fundamental type for creating the code we can see on the right side of the Recursion Creator. We will discuss `ElmExpr` in the following part. The `SF (Dict Int ElmExpr) StackFrame` links back to the previous `StackFrame`; it can be considered a recursive type.

For example, in Figure 7, for the first challenge in the Recursion Creator, the code on the right side is created by a `StackFrame` type

```

1 TopLevel
2   (Dict.fromList [
3     ( 1
4       , ExprListShape

```

```

5      (BracketList
6        [ ListFunApp ETInt
7          (ExprInt (IntConst (Just 1) ( 0, 255 ) 1 1))
8          (ETList ETShape)
9          ( "oneCircle", 2 )
10         |> Group
11       ]
12     )
13   ),
14   ( 2
15     , ExprLambda ETInt
16       (ETList ETShape)
17       { varName = "count"
18         , eyeState = 1
19         , bindIdx = 100
20         , elmExpr =
21           ExprListShape
22             (ListIf (IntGT (IntVar 100) (IntConst Nothing ( -10, 10 ) 1
23               0))
24               (BracketList
25                 [ Group
26                   (ListFunApp ETInt
27                     (ExprInt (IntSub (IntVar 100) (IntConst
28                       Nothing ( 1, 10 ) 1 1)))
29                     (ETList ETShape)
30                     ( "oneCircle", 2 )
31                   ]
32                 |> Move ( FloatConst (Just 4) ( -50, 50 ) 1 -5,
33                     FloatConst (Just 5) ( -50, 50 ) 1 0 )
34                   , Circle (FloatConst Nothing ( 0, 30 ) 1 5)
35                   |> Filled (RGB (FloatConst Nothing ( 0, 255 )
36                     10 255) (FloatConst Nothing ( 0, 255 ) 10 5) (FloatConst Nothing ( 0,
37                     255 ) 10 5))
38                 ]
39               )
40             )
41         (BracketList [])
42       )
43   }
44 )
45 ])
```

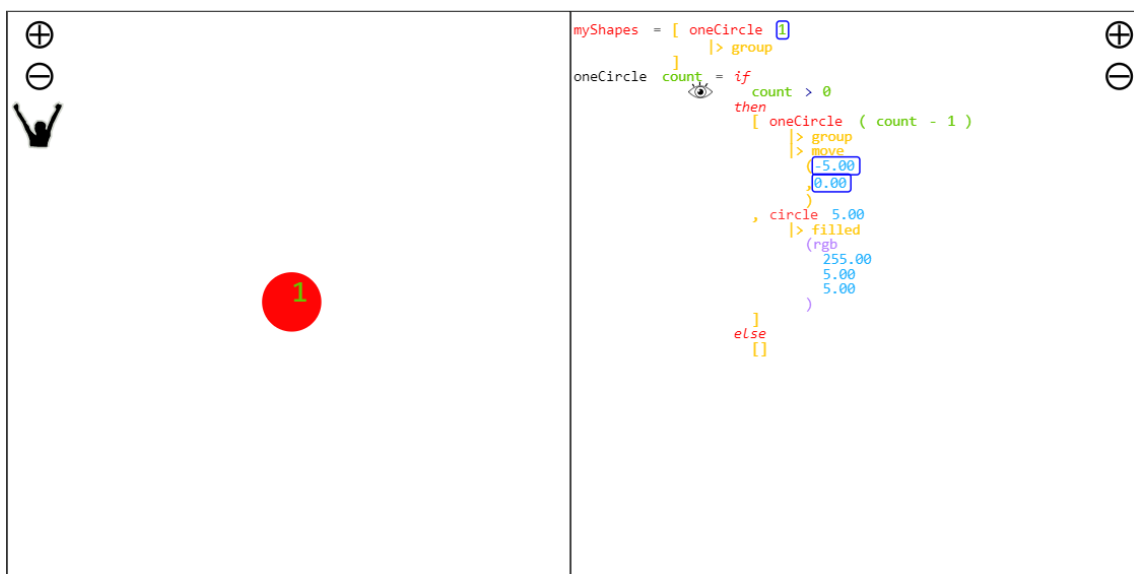


Figure 7: First One-Circle Challenge

The dictionary with key one, which mapped to an `ExprListShape`, created the

```
1 myShapes = [ oneCircle 1
2             |> group
3             ]
```

and the dictionary with the key two, which mapped to an `ExprLambda`, created the

```
1 oneCircle count = if
2     count > 0
3     then
4         [ oneCircle ( count - 1 )
5           |> group
6           |> move
7             (-5.00
8             ,0.00
9             )
10          , circle 5.00
11            |> filled
12              (rgb
13                255.00
14                5.00
15                5.00
16              )
17          ]
18     else
19         []
```

- `ElmExpr`

`ElmExpr` type can be considered a wrapper on basic Elm types like `Int` and `Float` and complicated types like `List` or function. Not all the variables in the code should be adjustable by students. In order to make some variables adjustable by the slider, but others are not, we have to create the `ElmExpr` type that wraps the basic Elm types with the controller (the blue box), then create a rendering module to `unboxElmExpr`. For example, one value `ElmExpr` type can have is `ExprInt ElmInt`, `ElmInt` is also a core type:

```
1 type ElmInt
2     = IntConst
3       (Maybe ControlIndex) -- Index for the control slider
4       ( Int, Int )          -- Allowed range
5       Int                  -- Step adjustment
6       Int                  -- Initial value
7     | IntVar VarRef
8     | IntAdd ElmInt ElmInt
9     | IntMult ElmInt ElmInt
10    | IntSub ElmInt ElmInt
```

An `ElmExpr` can look like

```
1 ExprInt (IntConst (Just 1)
2              ( 0, 255 )
3              1
4              2
5              )
```

which creates an adjustable `Int` variable with a default value of 2, with a range of 0 to 255, and each click on the slider would increase or decrease the variable by 1. All the other values that start with “`Expr`” are used for rendering as an adjustable variable. The values that start with “`Reduced`” are used for directly rendering as non-adjustable variables. They can be rendered as strings to display in the User Interface and copy-paste function later.

- ElmStencil

`ElmStencil` is the type of shape we can draw in the Recursion Creator. We only have six shapes: `Circle`, `Rectangle`, `Coral`, `Polygon`, `Ngon`, and `Text`. In the render module, the `ElmStencil` type will be rendered using the basic shape types we have in the library `GraphicSVG`.

For example, for an `ElmStencil` value `Circle (FloatConst (Just 25) (0, 30) 1 1)`, it would be rendered as a circle with the initial size set to 1, a blue slider box will be drawn on top of the size field to allow the user to change the size value. The range for the adjustable size is from 0 to 30, and the type is float.

- ElmShape

`ElmStencil` provides what shapes we can draw, and `ElmShape` controls how the shape would be drawn; for example, we can have a circle filled with a colour or an outline, and we can also rotate or move the shape. The last value `CaseListString (ElmList ElmString) ((VarName, BindIdx), (VarName, BindIdx), ElmShape) ElmShape` is for manipulating with string. Figure 8 demonstrates an example of manipulating string.

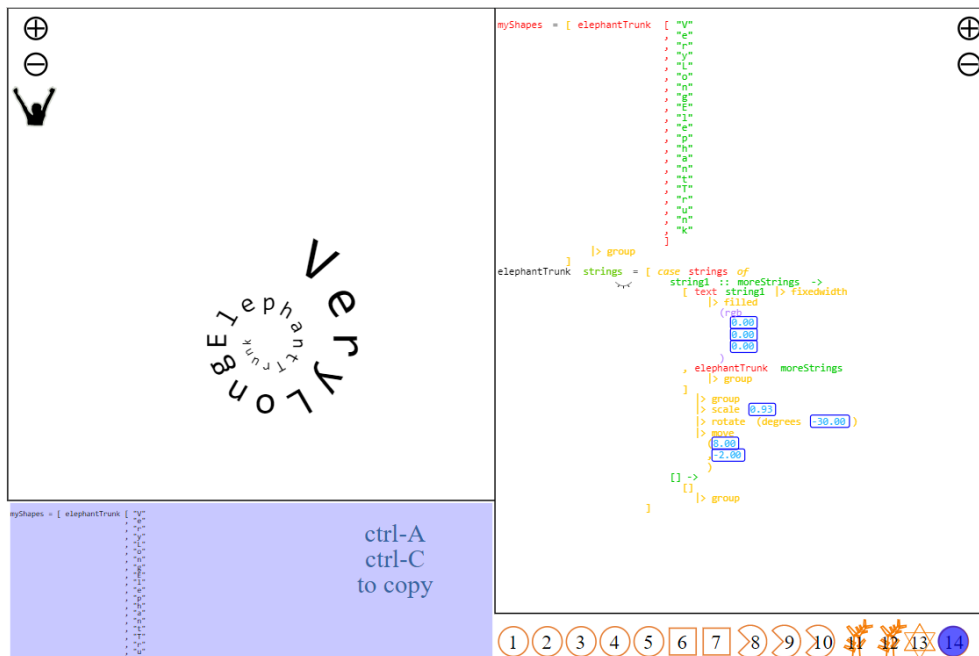


Figure 8: Recursive String Challenge

Other than the above self-created types, we also created types that control the slider

```

1 type ControlState
2   = Waiting
3   | Selected ControlIndex Adjustment Float
4   | Dragging ControlIndex Adjustment ( Float, Float ) ( Float, Float ) Float
5
6 type Adjustment
7   = AdjustFloat Float Float Float Float Float
8   | AdjustInt Int Int Int Int Int

```

`ControlState` type indicates the status of the slider: `Waiting`, `Selected` or `Dragging`, and the `Adjustment` type calculates how quickly the value should be changed.

2.2.2 Modules

In Recursion Creator, five modules play a vital role in creating interactive experiences for the students: the `AdjustConst` module, the `Copiable` module, the `HelperFunctions` module, the `Pretty` module and the `Render` module.

- **Render module**

The `Render` module is one of the essential modules. It is used to render all the shapes we can see on the left side of the UI. The `Render` module has multiple render functions to render different types.

`renderExpr` function is used to render an `ElmExpr` type, it has a signature

```
1 renderExpr : Model -> StackFrame -> ElmExpr -> ElmExpr
```

In this function, a `StackFrame` and an `ElmExpr` will be passed in as arguments, and then the function will return an updated `ElmExpr`. For different `ElmExpr`, it would call other different render functions in this module. For example, if the `ElmExpr` passed in is of `ExprInt`, it would call the `renderInt` function in this module, and if the payload is of type `ExprStencil`, then the function `renderStencil` would be called.

`renderListShape` function is used for rendering `ElmList`, the signature is

```
1 renderListShape : Model -> StackFrame -> ElmList ElmShape -> List (Shape  
    ElmViewMsg)
```

`ElmList` is also a self-created type, and it can have below values

```
1 type ElmList a  
2   = BracketList (List a)  
3   | ListPlus (ElmList a) (ElmList a)  
4   | ListFunApp  
5     ElmType  
6     ElmExpr  
7     -- input argument  
8     ElmType  
9     ( VarName, BindIdx )  
10    -- and a function  
11    | ListIf ElmBool (ElmList a) (ElmList a)  
12    | ListVar VarRef
```

For the `BracketList (List a)`, it would first render the list of unknown elements passed in and then enclose them with the `[]`. For the `ListPlus (ElmList a) (ElmList a)`, it is mainly functioning as combining two `ElmList` together. `ListFunApp` would provide a way to create our list of functions using `ElmExpr` we made before, `ListIf` would be rendered as the logic if, and `ListVar` would be rendered as an integer.

`renderShape` function has a signature as

```
1 renderShape : Model -> StackFrame -> ElmShape -> Shape ElmViewMsg
```

This function is used for rendering all the `ElmShape` in a `StackFrame`. As we mentioned in the `Core Types` section, the `ElmShape` type can determine how we want to draw the shapes, therefore in the `renderShape` function, each payload from `ElmShape` is mapped to the actual code in the `GraphicSVG` library. For example, if we have an `ElmShape` variable that looks like

```
1 Filled (RGB (FloatConst Nothing ( 0, 255 ) 10 255) (FloatConst Nothing ( 0,  
    255 ) 10 5) (FloatConst Nothing ( 0, 255 ) 10 5))
```


after several layers of unboxing, it would eventually be rendered as

```
1 filled (rgb 255.00 5.00 5.00)
```

which would be the valid compilable code in the GraphicSVG library.

`renderStencil` function has a signature as

```
1 renderStencil : StackFrame -> ElmStencil -> Stencil
```

This function renders the `ElmStencil` in a `stackFrame` into the regular stencil we used to create shapes in the GraphicSVG library.

- Pretty module

As crucial as the `Render` module is the `Pretty` module. All of the text and the slider panel for each variable on the right side of the UI are rendered using it. Similar to the `Render` module, in the `Pretty` module, we also have different functions targeting rendering different types. One big difference compared with the `Render` module is that for the function `prettyInt` and `prettyFloat` in the `Pretty` module, we would also create the blue slider box on top of the variable if the `ElmInt` passed in has a controller index as its payload.

`prettyInt` function has a signature as

```
1 prettyInt : Model -> Bool -> ElmInt -> ( Shape ElmViewMsg , Float , Float )
```

The `Bool` type value determines whether the integer will be closed in a bracket, and the `ElmInt` determines whether the integer will be an adjustable variable for the user in the UI. If the payload for `ElmInt` is an `IntVar VarRef`, OR an `IntConst (Maybe ControlIndex) (Int, Int) Int Int`, which has the controller index set as the type of `Nothing`, then it will be treated as a constant, and the heights for those will be constant. We will put a blue slider box on top for `ElmInt` with a payload with a controller index to make the value adjustable through the slider. For other `ElmInt` payload types such as `IntAdd`, `IntSub`, and `IntMult`, because the operands for them will be `IntVar VarRef`, OR `IntConst (Maybe ControlIndex) (Int, Int) Int Int`, for those types, they would recursively call the function `prettyInt` to unbox the value.

`prettyFloat` function has a similar structure; the only difference is that it would take an argument of an `ElmFloat` type.

The other important function in the `Pretty` module is the `prettyText` function. It has a signature as

```
1 prettyText : Color -> String -> ( Shape ElmViewMsg , Float , number )
```

`prettyText` takes in a string, and the colour we want for it then renders it as a `Shape` type in the GraphicSVG library, with its width and height.

After we have `prettyInt`, `prettyFloat`, and `prettyText` functions ready, all the other pretty functions, such as `prettyShape` and `prettyStencil`, utilize those three important functions to render the code itself to the UI as text.

- Copiable module

The `Copiable` module is similar to the `Pretty` module, but the implementation is less complicated. The fundamental function in this module is the function `copiableText`, which has a signature

```
1 copiableText : String -> Code
```

Code type here is just an alias for `List String`. This function maps a string into a list of copiable strings. Just as in the `Pretty` module, in the `Copiable` module, we also have functions `copiableInt` and `copiableFloat`. Those two functions utilize `copiableText` to output the variable value as a copiable text. For other functions such as `copiableStencil`, and `copiableShape`, they also recursively use `List copiableInt`, `copiableFloat` and `copiableText` to render the code for drawing shapes into copiable text.

- `AdjustConst` module

As we mentioned in previous chapters, students can use a slider to adjust the value of a variable. This `AdjustConst` module is created for updating the value after the adjustment.

This module contains a family of recursive functions which take the controller index, the adjusted value and the self-created `Elmxxx` types, such as `ElmInt`, `ElmFloat` or `ElmShape`, and return the updated `Elmxxx` type.

For example, the basic function to update `ElmInt` is the function `applyControlInt`, the signature is

```
1 applyControlInt : ControlIndex -> Adjustment -> ElmInt -> ElmInt
```

It would update the `ElmInt` with the new `Adjustment` value and then return that new `ElmInt`. All the other `applyControlxxx` functions, such as `applyControlShape`, `applyControlStencil` and `applyControlElmExpr`, would recursively call the `applyControlInt` function or `applyControlFloat` function to update the value.

- `HelperFunctions` module

As suggested in the module name, `HelperFunctions` contains all the helper functions and shapes we used in the Recursion Creator UI. The “VictoryGesture Button,” “Zoom-in button,” “Zoom-out Button,” and all the challenge outlines are implemented in this module.

The container for all the shapes we have is an `Array of (String, number, ElmExpr)`, `String` is for the shape’s name, `number` is for the index, and `ElmExpr` is the actual code that draws the shape. For example, the code for the first challenge is

```
1 ( "oneCircle"
2   , 2
3   , ExprLambda ETInt
4     (ETList ETShape)
5     { varName = "count"
6       , eyeState = 1
7       , bindIdx = 100
8       , elmExpr =
9         ExprListShape
10        (ListIf (IntGT (IntVar 100) (IntConst Nothing ( -10, 10 ) 1
11          0))
12              (BracketList
13                [ Group
14                  (ListFunApp ETInt
15                    (ExprInt (IntSub (IntVar 100) (IntConst
16                      Nothing ( 1, 10 ) 1 1)))
17                    -- input argument
18                    (ETList ETShape)
19                    ( "oneCircle", 2 )
20                  )
21                  |> Move ( FloatConst (Just 4) ( -50, 50 ) 1 -5,
22                        FloatConst (Just 5) ( -50, 50 ) 1 0 )
```

```

20         , Circle (FloatConst Nothing ( 0, 30 ) 1 5)
21         |> Filled (RGB (FloatConst Nothing ( 0, 255 )
10 255) (FloatConst Nothing ( 0, 255 ) 10 5) (FloatConst Nothing ( 0,
255 ) 10 5))
22     ]
23 )
24 (BracketList [])
25 )
26 }
27 )

```

2.2.3 Code Architecture

The Elm architecture essentially can be divided into three parts: Model, View and Update. Similar to the typical MVC architecture, instead of a Controller, in Elm, we use Update to play the role of a Controller. Data are stored in the Model, and the View is the User Interface users can see. When the user changes the value in the View, it updates the Model through the Update method instead of directly modifying the model. As explained by [Poudel n.d.], the Elm runtime would initialize and render the view with the initial model. If the user triggers any messages in the view, the Update part will send the message to the current model and then return with an updated model, and last, render the view again with the latest updated model. The process mentioned in [Poudel n.d.] is depicted in Figure 9. In Recursion Creator, we also adopted the Model, View and Update

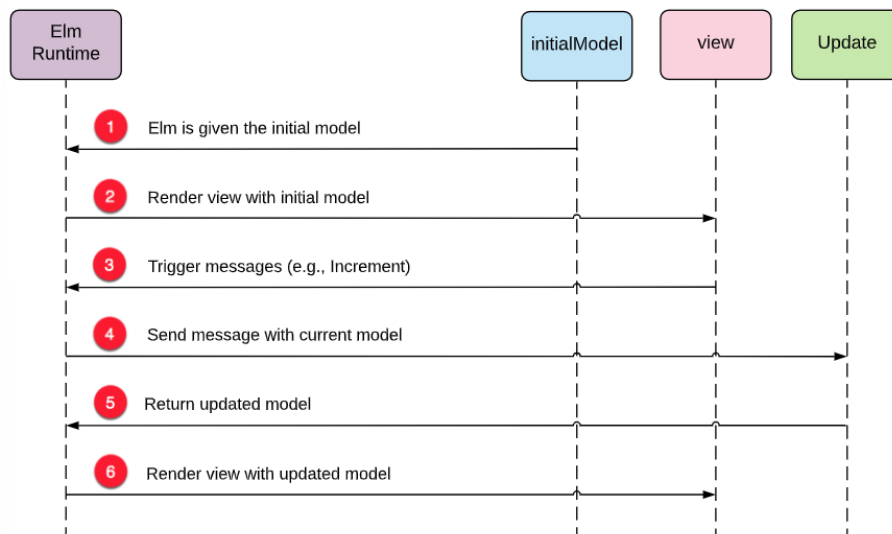


Figure 9: How the Elm Runtime Interacts with Model, View, and Update

mechanism. The model was initialized with the below settings

```

1 init =
2   { time = 0
3     , controlState = Waiting
4     , topLevel =
5       TopLevel
6         (testTopLevel |> List.map (\( _, idx, expr ) -> ( idx, expr ))
7           |> Dict.fromList)
8     , variables = mkBindingDict testTopLevel
9     , exampleoutline = "oneCircle"

```

```

10   , buttonHighlights = "oneCircle"
11   , codeScale = 0.5
12   , shapeScale = 1
13   , outlineshape = False
14   }

```

The `testTopLevel` is of type `StackFrame`, which contains a list of two tuples of type `(VarName , BindIdx, ElmExpr)`. The most important object is the `ElmExpr` which constructs the base shape we want to draw. The `controlState` is set for the slider's state. The default state for all the sliders on the adjustable variable is "waiting." The `variables` is a type of `Dict Int (String, Color)`, which will be used in the `Copiable` module to decide the colour in which a variable would be rendered. `exampleoutline` and `buttonHighlights` are of type string, `exampleoutline` determines which shape should have the outline as default, and `buttonHighlights` highlights the button on the right corner of the Recursion Creator as the default view. `codeScale` and `shapeScale` are the scales of the shape on the left and the code on the right, respectively. The last `outlineshape` is to decide if the challenge mode is enabled as default.

The view is initialized as

```

1 view : Model -> Collage ElmViewMsg
2 view model = collage 192 128 (myShapes model)

```

`collage` is the canvas created in the `GraphicSVG` library, and `myShapes` takes in the model and returns `List (Shape ElmViewMsg)` for the `GraphicSVG` library to render on the canvas.

The update function takes an `ElmViewMsg` and `Model` as arguments and maps the `ElmViewMsg` to each `applyControl` function in the `AdjustConst` module to update the values in the `Model`, then return the updated `Model` to the runtime.

As shown in Figure 10, the `Render` module, `Pretty` module and `Copiable` module are used on the view to render the shapes, the code the user can adjust, and the code users can copy/paste. Models are constructed using the self-created types; the essential structure is the `StackFrame`. In the update, each `ElmViewMsg` is mapped to update functions in the `AdjustConst` Module.

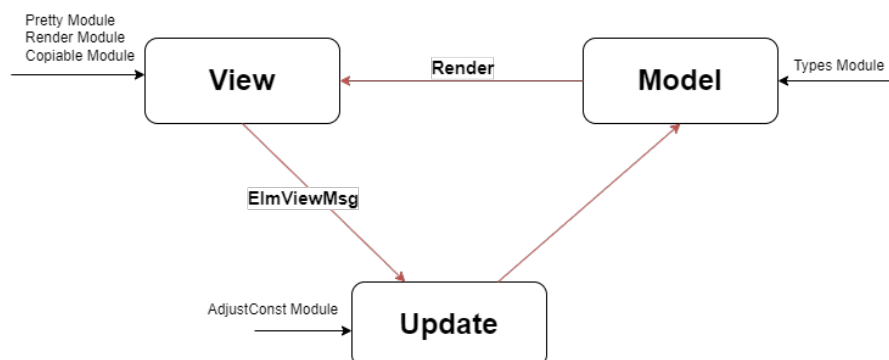


Figure 10: MVU for Recursion Creator

3 Recursion Creator Camp

In the summer of 2022, we conducted a three-day summer camp for high school students with a certain amount of Elm programming experience. The camp utilized the Recursion Creator tool to teach Recursion to the students.

3.1 Curriculum

The camp started with those recursive shapes, as shown in Figure 11 that we can find in our daily life as well as in art pieces. Starting with real-life recursive examples could encourage students' curiosity to explore Recursion further. To ensure all the students

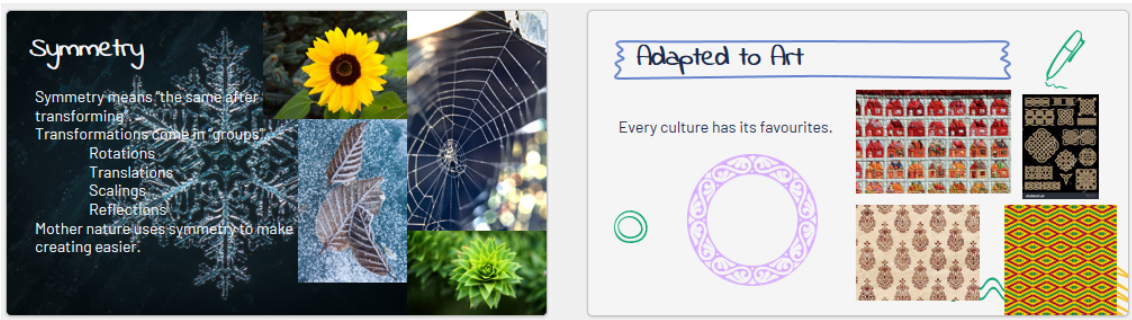


Figure 11: Recursive Shapes Examples

have fresh memory about Elm and have a clear understanding of the Recursive concept, we divided the curriculum into three sessions.

1. Review Elm Programming
2. Introduce Recursion
3. Recursion Shape Challenge

3.1.1 Review of Elm programming

All the students had 1 to 4 years of Elm programming experience, and they could use Elm and the library GraphicSVG to draw basic and complex shapes. Nevertheless, in order to ensure each student has solid programming knowledge to draw the recursive shapes in the following sessions, we provided a brief review of Elm programming at the beginning of the camp.

The review of Elm covers all the basic Elm syntax we use to draw shapes, the concept of a function and the logical syntax if...else. All the basic programming is taught along with examples, as shown in Figure 12. In order to introduce the concept of “Function” to students more intuitively, we adopted the analogy of “steps to make a pie.” Figure 13 provides detailed instructions. It includes step-by-step instructions on how to make a pie on the left-hand side, while the right-hand side displays the Elm code that explains how the function groups the steps together.

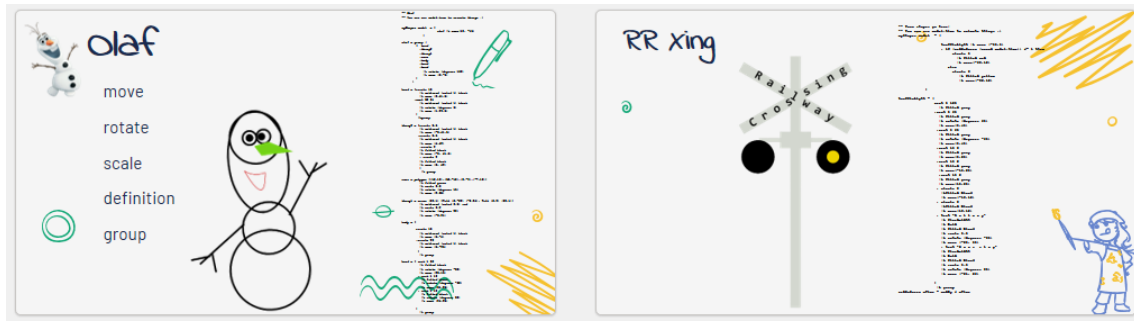


Figure 12: Basic Elm



Figure 13: Introduce Function

3.1.2 Introduce Recursion

To start the introduction of Recursion in an easier-to-understand way, we started with a recursive bedtime story.

“Once upon a time, there was a child who could not sleep, so her mother told her a story about a little frog, who could not sleep, so the frog’s mother told her a story about a little bird, who could not sleep, so the bird’s mother told her a story about a little puppy, and the little puppy fell asleep, and the little frog fell asleep, and the little frog fell asleep, and the child got hungry after such a long story!”

Students are asked to use the Elm programming tool MacOutreach.Rocks to draw out the story in an animation style using their own created animals, as shown in Figure 14.

3.1.3 Recursion shape challenge

After introducing students to the basic concept of Recursion, we briefly introduced the Recursion Creator and how to use the tool. Before introducing students to the Recursion Shape Challenges, we included a new game - “Divide Conquer Combine.”

“Divide Conquer Combine” aims to introduce the concept of breaking up a big problem

Once upon a time, there was a child
who couldn't sleep, so her mother told her a story about a little frog,
who couldn't sleep, so the frog's mother told her a story about a little bird,
who couldn't sleep, so the bird's mother told her a story about a little puppy,

and the little puppy fell asleep,
and the little bird fell asleep,
and the little frog fell asleep,
and the child got hungry after such a long story!



Figure 14: Recursive bedtime story

into small problems, finding solutions to the small problems, and then combining the small solutions into a bigger solution, and, last but not least, stopping the process.

The last part is the Recursion Shape Challenges mentioned in Chapter Two. Students are divided into groups, and each group is asked to finish all the shape challenges in Recursion Creator Tool.

3.1.4 Assessment

At the end of each day, we provide a Self Check table for each group to measure students' understanding of the materials. Each member was asked a programming question related to the materials we covered that day, and they were asked to write the code for the question and share the code. Other members in the same group will be the reviewer for the answer.

4 Conclusion

Recursion could be a challenging topic in our programming-learning path due to

- The lack of recursive examples in our physical life
- Difficulties in tracing or debugging recursion
- Misconceptions about the base case

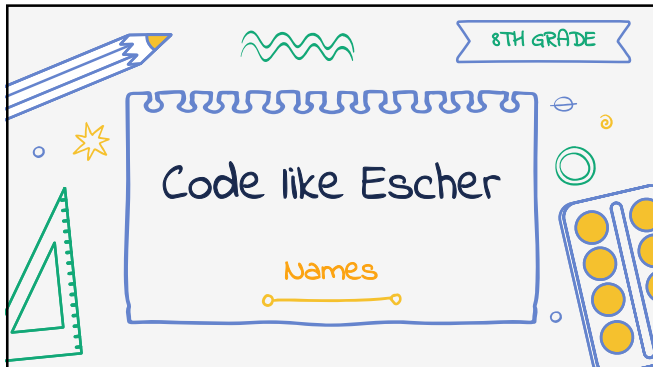
However, Recursion Creator can assist in teaching Recursion by minimizing the above difficulties. Recursion Creator uses different recursive shapes found in our physical life as examples; from the visual aspect, this could help increase students' interest in learning Recursion. Additionally, Recursion Creator provides an interface to trace the recursive part by changing several variables, and students can directly trace the changes in the shapes they draw. The base shape in Recursion Creator provides a new point of view about the base case in a Recursion, combined with some other knowledge sharing on the base case; this could help students have a more solid understanding of the base case of a Recursion.

For future work, we would like to

- Develop an assessment system for the learning result, so we can quantitatively analyze students' improvement in understanding Recursion;
- Conduct additional research on teaching Recursion in order to add more evidence-based features in the Recursion Creator to improve students' learning experiences; and
- Fix bugs in current Recursion Creator.
- Upgrade the Recursion Creator to a Maze Game where students can explore each branch recursively. The goal of the game is to let students figure out what shape the maze is by exploring all the branches. The game would provide a mechanism allowing the player to locate where they are by zooming in and out, which would act as tracing a recursive call in Recursion.

5 Appendix - Summer Camp Slides

The slides used in the Summer Camp are attached below



1



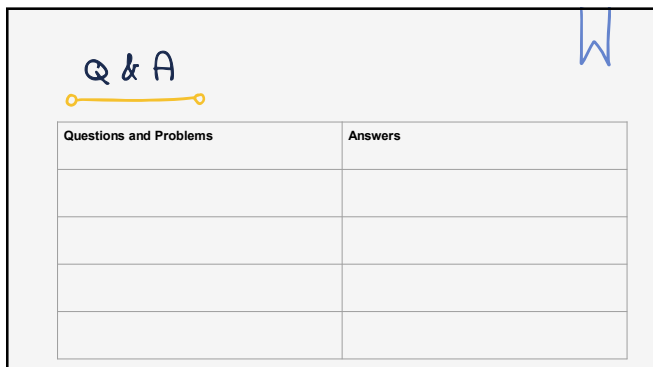
2



3



4



5



6

[illegible][illegible]

Function

1. Make the pastry
2. Tip **myFilling** on the pastry
3. Add the topping
4. Bake it and Smile :)

```
-- Your shapes go here!
-- You can use model.time to animate things :)
myShapes model =
  [
    make_AmericanPie blueberryFilling
  ]

--These are the definitions for the elements in a pie
myPastry = circle 20
  > filled lightBrown

appleFilling = circle 25 > filled red

blueberryFilling = circle 25 > filled blue

mushroomTop = [
  rect 50 3 > filled lightBrown
  , rect 50 3 > filled lightBrown > move(0,10)
  , rect 30 3 > filled lightBrown > move(0,-10)
  , rect 40 3 > filled lightBrown > move(0,20)
  , rect 40 3 > filled lightBrown > move(0,-20)
  , rect 3 50 > filled lightBrown > move(10,0)
  , rect 3 50 > filled lightBrown > move(-10,0)
  , rect 3 40 > filled lightBrown > move(20,0)
  , rect 3 40 > filled lightBrown > move(-20,0)
]
> group

--This function will make a delicious pie
make_AmericanPie myFilling = [myPastry
  , myFilling
  , mushroomTop
]
> group
```



Self check

Member Name	Question for ya!	ShareLink	Verified by
Jack	What should you do, if Olaf wants a square nose?	https://macosteach.rockstarherald.com/5612e3a8	
	What should you do, if the railway crossing wants a Pink & Blue lights?		
	What should you do, if Olaf wants stronger arms?		
	What should you do, if the railway crossing wants one square light and one triangle light?		
	What should you do, if you want to eat a ngon pie?		

[illegible]

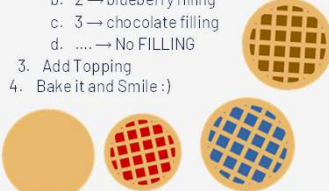
Self Check

Member Name	Question for ya!	ShareLink	Verified By:
Jack 	What should you do, if you want a blue pie with star-shaped toppings on it?	https://macosfreak.rockstargames.com/shares/6f42e3a1a1c	   
	What should you do, if you want a yellow pie with bar-shaped toppings on it?		
	What should you do, if you want a pink pie with orange triangle toppings on it?		
	What should you do, if you want a ngn pie with circle shape toppings on it?		
	What should you do, if you want a square pie with ngon toppings on it?		

Making pie with case...

make_AmericanPieWithCase FillingChoice

1. Make the pastry
2. Choose Filling using FillingChoice
 - a. 1 → apple filling
 - b. 2 → blueberry filling
 - c. 3 → chocolate filling
 - d. → No FILLING
3. Add Topping
4. Bake it and Smile :)



```

-- Your shapes go here!
-- You can use model.time to animate things in
myShape = circle 20
[
  make_AmericanPieWithCase 4
]

-- These are the definitions for the elements of a pie
myPastry = circle 20
  > filled lightbrown
  > filled lightbrown

appleFilling = circle 25 > filled red
blueberryFilling = circle 25 > filled blue
chocolateFilling = circle 25 > filled darkbrown
noFilling = circle 25 > filled lightbrown

regularTop = [
  rect 50 3 > filled lightbrown
  , rect 50 3 > filled lightbrown > move(0,10)
  , rect 50 3 > filled lightbrown > move(0,-10)
  , rect 40 3 > filled lightbrown > move(0,20)
  , rect 40 3 > filled lightbrown > move(0,-20)
  , rect 3 50 > filled lightbrown > move(10,0)
  , rect 3 50 > filled lightbrown > move(-10,0)
  , rect 3 50 > filled lightbrown > move(25,0)
  , rect 3 40 > filled lightbrown > move(-25,0)
  ]
  > group

--This function will make a delicious pie
make_AmericanPieWithCase whichFilling = myPastry
make_AmericanPieWithCase whichFilling = myPastry
  , case whichFilling of
    1 -> appleFilling
    2 -> blueberryFilling
    3 -> chocolateFilling
    _ -> noFilling
  , regularTop
  > group

```

13

Self Check

Member Name	Question for ya!	ShareLink	Verified By
Jack	What should you do, if you want your pie to have apple filling with triangle shaped toppings on it?	https://masoudreazach.rocks/shares/5542a3a5a5	
	What should you do, if you want your pie to have chocolate filling with star shaped toppings on it?		
	What should you do, if you want your pie with no filling but with ngon shaped toppings on it?		
	What should you do, if you want your pie to have strawberry filling with triangle shaped toppings on it?		
	What should you do, if you want your pie to have blueberry filling with circle shaped toppings on it?		

14



Break

15



Let's make your own animals !!

16


Self Check

Member Name	PasteBin Link	Verified By

17

Answers

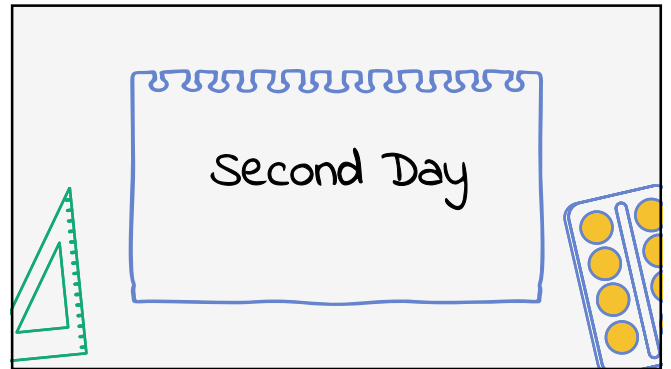
- 1.
- 2.



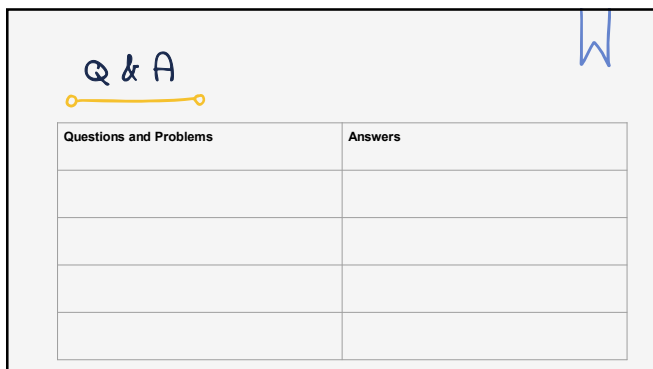
18



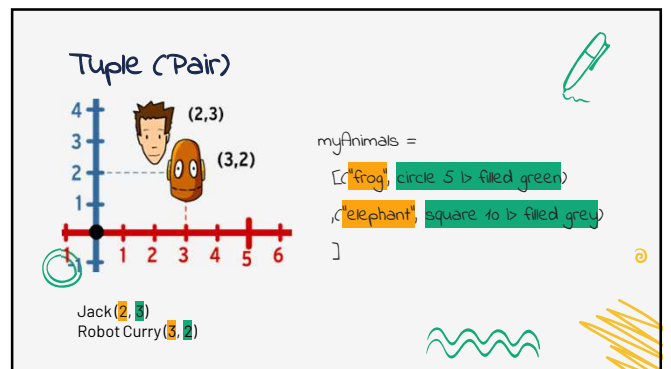
19



20



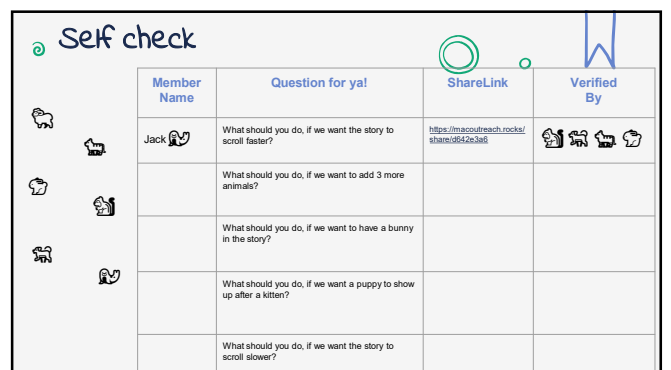
21



22



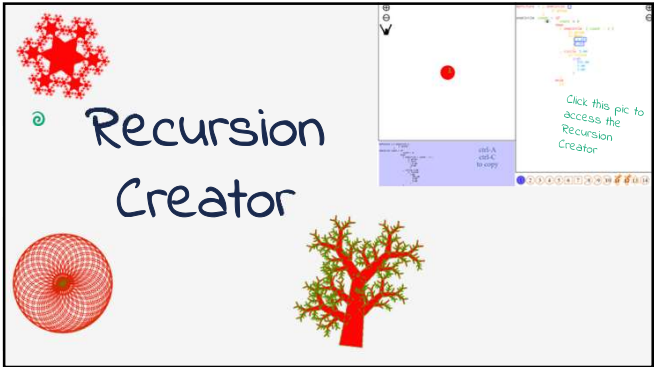
23



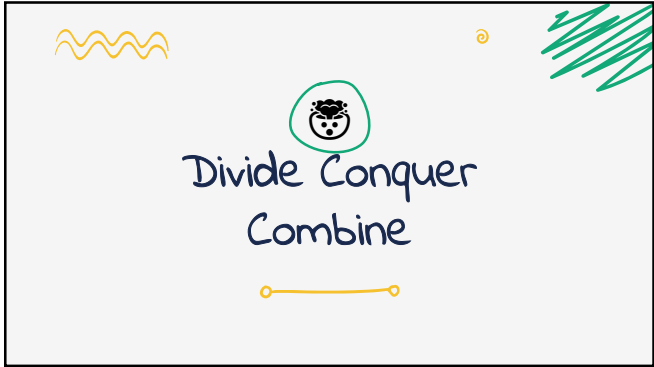
24



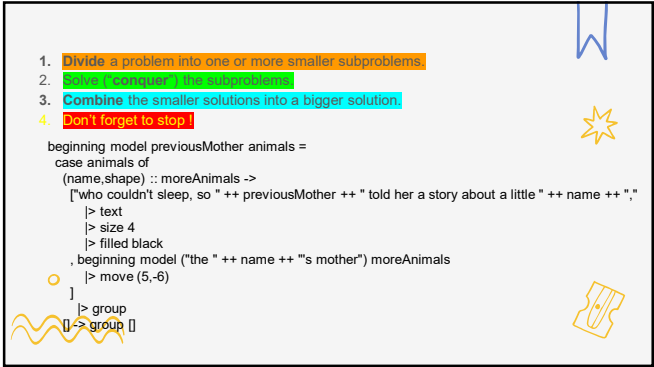
25



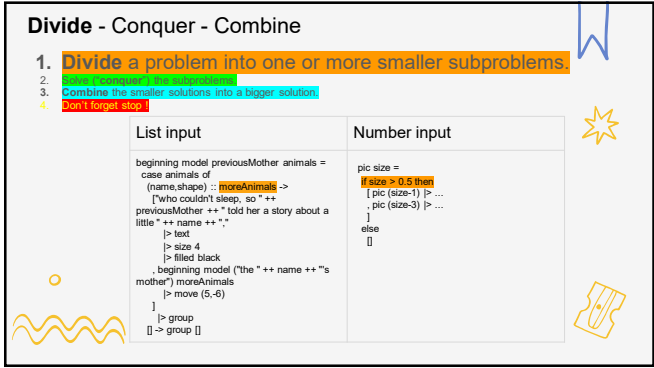
26



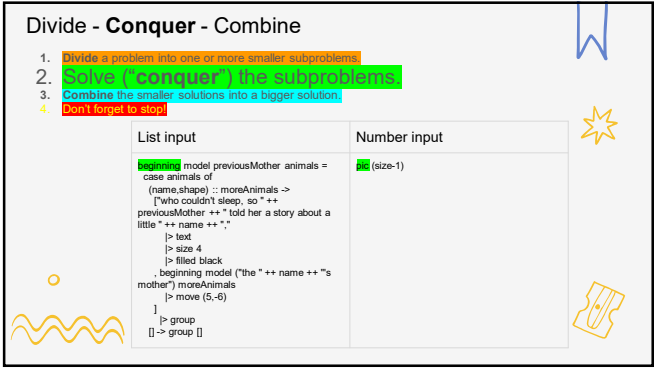
27



28



29



30

Divide - Conquer - Combine

1. Divide a problem into one or more smaller subproblems.

2. Solve the subproblems.

3. Combine the smaller solutions into a bigger solution.

4. Don't forget to stop!

List input	Number input
<pre>beginning model previousMother animals = case animals of (name,shape) :: moreAnimals -> [who couldn't sleep, so " ++ previousMother ++ " told her a story about a little" ++ name ++ " > text > size 4 > filled black ,beginning model ("the " ++ name ++ "'s mother") moreAnimals > move (5,-6)] > group [] -> group []</pre>	<pre>[...] > text pic (size - 1)] > group</pre>

31

Divide - Conquer - Combine

1. Divide a problem into one or more smaller subproblems.

2. Solve the subproblems.

3. Combine the smaller solutions into a bigger solution.

4. Don't forget to stop!

List input	Number input
<pre>beginning model previousMother animals = case animals of (name,shape) :: moreAnimals -> [who couldn't sleep, so " ++ previousMother ++ " told her a story about a little" ++ name ++ " > text > size 4 > filled black ,beginning model ("the " ++ name ++ "'s mother") moreAnimals > move (5,-6)] > group [] -> group []</pre>	<pre>if size > 0.5 then [...]] else []</pre>

32

SEE YOU
LATER
-FRENCHTOP

Break

33

Shape
challenge

34

Self Check

Member Name

PasteBin Link

Verified By

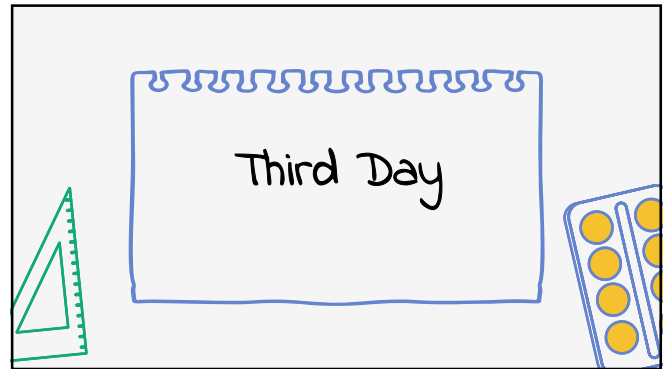
35

Answers

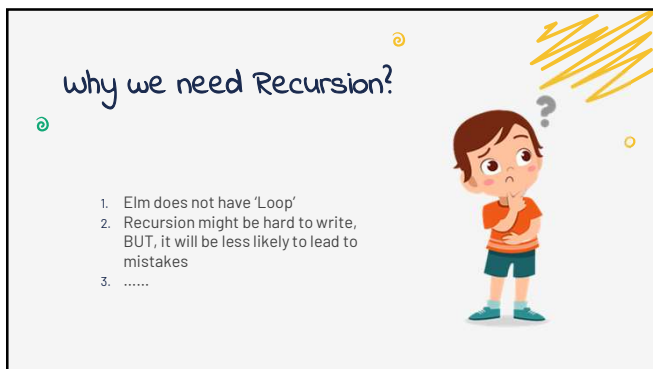
36



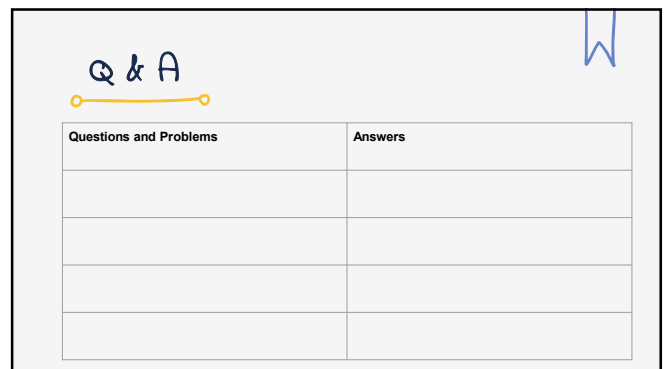
37



38



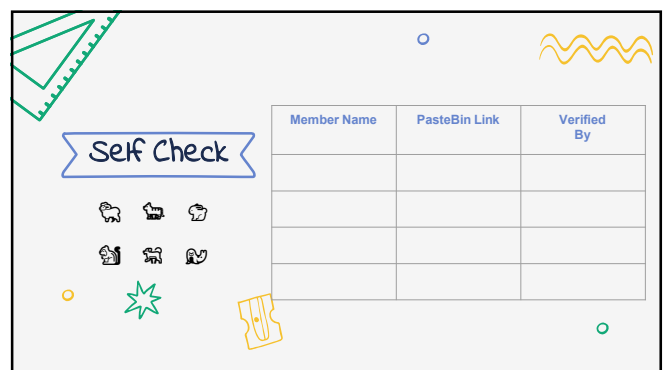
39



40



41




42


Coral Challenge

In your group,

1. Find pictures of coral
2. Discuss how symmetry appears in the different coral.
3. Pick one coral each to draw.
4. Assemble.
5. Present solution.



43



Break

44

Self Check

Member Name	PasteBin Link	Verified By

45

Quick review

```

graph TD
    Recursion[Recursion] --> Elm[Elm]
    Recursion --> RecursionCreator[Recursion Creator]
    Recursion --> TreeCoral[Tree & Coral]
    Elm --- ElmDesc[Group, function, if...else]
    RecursionCreator --- RecursionCreatorDesc[Draw shapes using recursion]
    TreeCoral --- TreeCoralDesc[Using recursion]
  
```

46

Answers

- 1.
- 2.

47

More Recursion Examples:

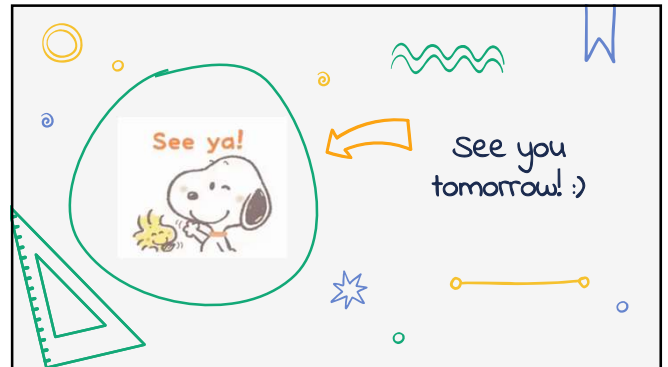
<https://gist.github.com/CSchank/26596747f572921c256656d8a2ba95ab>



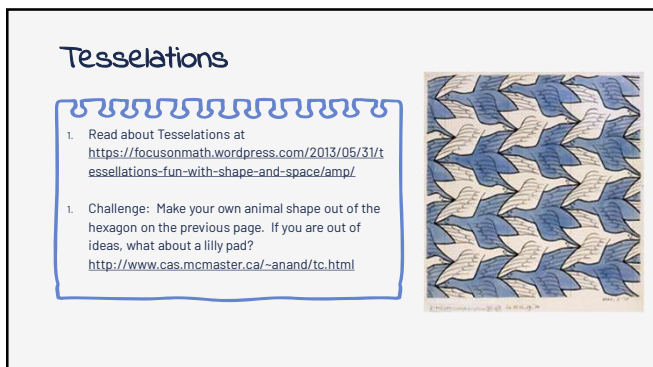
48



49



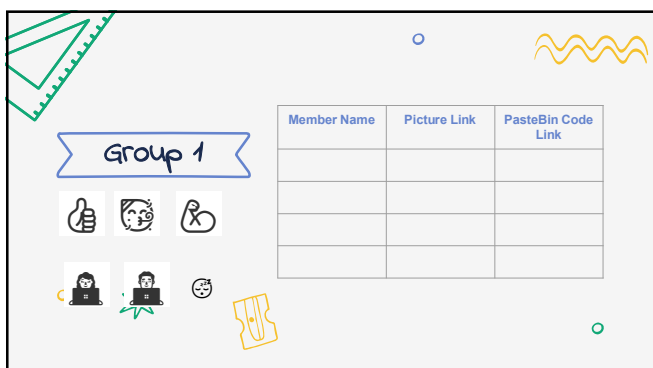
50



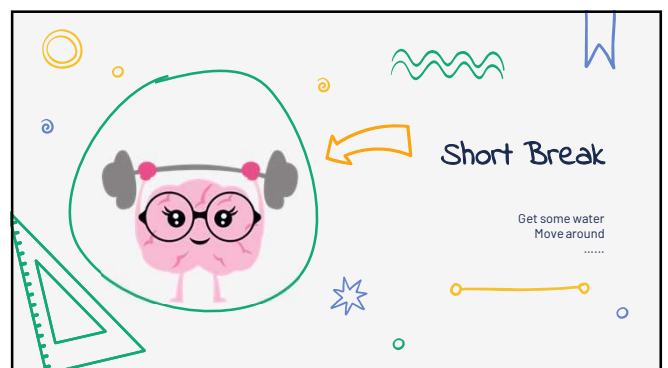
51



52



53



54

References

- Benander, Alan C, Barbara A Benander, and Howard Pu (1996). "Recursion vs. iteration: An empirical study of comprehension". In: *Journal of Systems and Software* 32.1, pp. 73–82.
- Booth, S (1992). "The experience of learning to program. Example: recursion". In: *Proceedings of the 5th Workshop of the Psychology of Programming Interest Group*, pp. 122–145.
- Close, John and Darina Dicheva (1997). "Misconceptions in recursion: diagnostic teaching". In: *Proceedings of the sixth eurologo conference "learning and exploring with logo*, pp. 132–140.
- Daylight, Edgar G (2011). "Dijkstra's rallying cry for generalization: The advent of the recursive procedure, late 1950s–early 1960s". In: *The Computer Journal* 54.11, pp. 1756–1772.
- Dicheva, Darina and John Close (1996). "Mental models of recursion". In: *Journal of Educational Computing Research* 14.1, pp. 1–23.
- Gal-Ezer, Judith and David Harel (1998). "What (else) should CS educators know?" In: *Communications of the ACM* 41.9, pp. 77–84.
- Haberman, Bruria and Haim Averbuch (2002). "The case of base cases: Why are they so difficult to recognize? Student difficulties with recursion". In: *Proceedings of the 7th annual conference on innovation and technology in computer science education*, pp. 84–88.
- Kessler, Claudius M and John R Anderson (1986). "Learning flow of control: Recursive and iterative procedures". In: *Human-Computer Interaction* 2.2, pp. 135–166.
- Lee, Okhwa and Richard Lehrer (1988). "Conjectures concerning the origins of misconceptions in LOGO". In: *Journal of Educational Computing Research* 4.1, pp. 87–105.
- McCauley, Renée et al. (2015). "Teaching and learning recursive programming: a review of the research literature". In: *Computer Science Education* 25.1, pp. 37–66.
- Poudel, Pawan (n.d.). *Beginning Elm: a gentle introduction to Elm programming language*. URL: <https://elmprogramming.com/>.
- Roberts, Eric and Eric Roberts (1986). *Thinking recursively*. J. Wiley.
- Segal, Judith (1994). "Empirical studies of functional programming learners evaluating recursive functions". In: *Instructional Science* 22.5, pp. 385–411.