Practice CS Concepts with Snap!

A companion to "Learn CS Concepts with Snap"
(2nd edition)

With lots of additional Review Questions and Programming Problems

Abhay B. Joshi

Book series on "Learning computer programming and CS principles"

To Richard Feynman,
a teacher who could make complex ideas accessible to everyone

All rights reserved by the author

Published by:
SPARK Institute and Publications
3554 - 173rd CT NE
Redmond, WA 98052, USA
1 June 2019

Cover design by:
Ravindra Pande

To order your digital or printed copy:
Go to Amazon.com

If print version not available in your country:
Write to: abjoshi@yahoo.com
Pricing and other info: http://www.abhayjoshi.net/spark/bsnaps.pdf

Other books in this series:
Advanced Scratch Programming
Pen Art in Snap Programming

Click http://www.abhayjoshi.net/spark/csseries.pdf to see the complete list.

Preface

We must be very careful when we give advice to younger people; sometimes they follow it!
– Edsger W. Dijkstra

Background

This book is a companion to the book titled "Learn CS Concepts with Snap" by the same author. It provides additional review questions to evaluate your understanding of CS concepts, and additional programming assignments to practice all these concepts.

The idea of using *computer programming as a medium for learning* is rapidly gaining acceptance. The benefits of learning programming and *computer science* concepts well before college – even in elementary grades – are well-understood.

Here is a list of some of the amazing things that happen when children and young adults engage in computer programming:

- They become *active* and *creative* learners, because they explore ideas through a hands-on activity with an infinitely powerful tool.
- They learn to think about and analyze *their own thinking*, because that is the only way to program computers.
- They learn to solve complex problems by breaking them into smaller sub-problems.
- They learn a new way of thinking (called "computational" thinking).
- In the world of programming, answers are not simply "right" or "wrong"; this prepares a student's mindset for real-life problems.
- The learning process is transformed from *acquiring facts* to *thinking creatively and analytically*.

How the book is organized

The book is organized exactly along the lines of the original book: as a series of units – each containing a bunch of CS concepts and associated programming activities. Every programming assignment requires understanding of all the concepts presented in the current as well as all preceding units.

Answers to all review questions and links to working programs for most of the programming exercises in the book are available online. Download them at: http://www.abhayjoshi.net/spark/snap/book0s/solutions.pdf.

Hardware and Software

You can do all your Snap programming work online by creating your own account at http://snap.berkeley.edu.

Acknowledgements

I wish to thank Tanuja Joshi for reviewing the material of this book diligently and tirelessly and for providing me with valuable suggestions. I wish to thank Ravindra Pande for creating a truly beautiful cover for the book.

I wish to thank TEALS (https://www.tealsk12.org) – a nonprofit organization dedicated to the cause of teaching computer science to all high school students – for allowing me to teach Snap using their remote classroom infrastructure.

Finally, this book would not have been possible without the constant encouragement of my friends and family.

I do hope that you will find this book useful and enjoyable.

Abhay B. Joshi (abjoshi@yahoo.com)
Seattle, USA
1 June 2019

Author's background

As a freelance teacher, Abhay's area of interest is teaching Computer Programming as "the exciting new magic" and also as a "medium for learning" in the constructivist tradition. He has been teaching regularly to elementary, middle, and high school students in WA, USA and Pune, India since 2008. He teaches at Aksharnandan School in Pune, India every summer, and works with TEALS (http://tealsk12.org/) to teach Computer Science to high school students in the US. Abhay teaches both in person and over the Internet using video-conferencing tools. Since 2011, he has authored or co-authored several programming books, which are based on Scratch, Snap, and Logo. He has written several articles to promote CS education, and offers teacher-training workshops to encourage aspiring teachers to experiment with this idea.

Abhay has been associated with the Software Industry since 1988 as a programmer, product developer, entrepreneur, and teacher. After getting an MS in Computer Engineering from Syracuse University (USA), he worked as a software engineer for product companies that developed operating systems, network protocols, and secure software. In 1997, Abhay co-founded Disha Technologies, which grew to be a successful software services organization.

Programming remains one of Abhay's favorite hobbies, and he continues to explore the "entertaining, intellectual, and educational" aspects of programming.

Creative activities bring joy, meaning, and purpose to people's lives. Focusing on creativity is not just an economic imperative, it is also a path towards human fulfillment, an opportunity for humans to be more human. – Mitchel Resnick, MIT

Contents

Background 3
Unit 0: Introduction 11
Unit 1: First Project in Snap 13
Unit 2: Interactive Animation 31
Unit 3: First Game Project 39
Unit 4: Advanced Game 47
Unit 5: Power of Variables 61
Unit 6: Clones 85
Unit 7: Procedures and Pen Art 95
Unit 8: Additional CS Concepts 125
Appendix A: Program Demos 145

Unit 0: Introduction

Learning to program can change the way children learn everything else. – Seymour Papert

The purpose of this unit is only to explain to you the format of each subsequent Unit in this book. Each Unit will contain sections listed below.

Concepts practiced in this Unit:

In this section, we list the Snap and CS concepts relevant to this Unit. The review questions and programming assignments are based on these concepts and on those covered in all previous Units.

Review questions:

In this section, you will find questions that will help you review your understanding of the concepts learnt thus far. This understanding is necessary before you go about solving the programming assignments that follow.

Programming problems:

In this section, you will find programming problems that will help you practice your understanding of all the Snap and CS concepts you have learnt in this Unit and all the preceding Units.

Appendix A provides two pieces of information that may come handy when you tackle these programming assignments:

One is links to demo videos that demonstrate the behavior of each program. Of course, these are only sample demos, and you can be totally creative with your

own programs. The other is a file that tells you which Snap/CS concepts are relevant for each program.

Notes and guidelines for programming problems:

In this section, you will find guidelines and hints that might help you with the programming problems. In some cases, you will also find possible designs and algorithms. Use this section only if needed!

Unit 1: First Project in Snap

The language in which you'll spend most of your working life hasn't been invented yet, so we can't teach it to you. Instead we have to give you the skills you need to learn new languages as they appear. – Brian Harvey, University of California at Berkeley

Concepts practiced in this Unit

- Snap User Interface
- Paint editor
- Sequence
- Motion commands
- Simple looping (repeat, forever)
- Absolute motion
- Relative motion
- Smooth motion using repeat
- Nested looping
- XY geometry
- Costume-based animation
- Multiple backdrops
- 3-D effect using repeat
- Basic sound commands
- Synchronization using broadcasting

Concept Review Questions:

1. The actors in Snap programs are known as:
   A. Turtles
   B. Sprites
   C. Costumes
   D. All of the above

2. What will the following command do?

   A. Turn counter-clockwise 90 degrees.
   B. Point in the East direction.
   C. Turn clockwise 90 degrees.
   D. Increase the temperature to 90 degrees.

3. If a sprite is staring in some unknown direction, which block below will "straighten" it up?

   A.

   B.

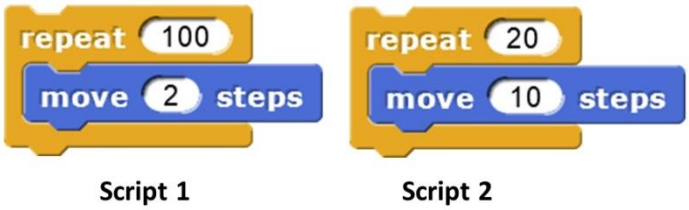
   C.

   D.

4. When you run the command

   A. The sprite moves only if it is at the center
   B. The sprite takes 300 jumps
   C. The sprite slides slowly until it has covered 300 steps
   D. The sprite jumps forward instantly 300 steps

5. See the following scripts. Which script will make the sprite move faster?



   A. Script 1
   B. Both go the same speed
   C. Script 2
   D. Sometimes Script 1, sometimes Script 2

6. The sprite has 3 costumes, and has the following script.

   A. After showing the 3 costumes the script will stop.
   B. The "wait" command will have no effect on the animation.
   C. The sprite will cycle through its costumes continuously.
   D. The costumes will appear at different places on the screen.

7. See the following program. The Sprite takes 8 seconds to complete the program.

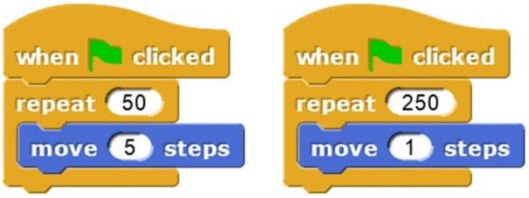
   If you change the move command to "move 2 steps", how long will program take?

A. 14 seconds
B. 5 seconds
C. 8 seconds
D. Impossible to tell

8. In the instruction

the "bounce" effect is equivalent to:
   A. Turn right 180 degrees
   B. Turn right 360 degrees
   C. Turn right 90 degrees
   D. Turn left 90 degrees

9. See the two scripts below and decide which statement is true.



   A. First script goes a shorter distance.
   B. First script will move the sprite faster.
   C. Second script will move the sprite faster.
   D. Second script goes a shorter distance.

10. Can the following script be made shorter using REPEAT? If yes, how many total command blocks will the new script have?

A. Yes, the number of blocks will be 8
B. Yes, the number of blocks will be 6
C. Yes, the number of blocks will be 5
D. No, the script cannot be made shorter.

11. If the sprite is at the center, and you want to move it to (-200, -200) you should:
A. first move it 200 steps north and then 200 steps south
B. first move it 200 steps south and then 200 steps west
C. first move it 200 steps west and then 200 steps north
D. first move it 200 steps east and then 200 steps south

12. My sprite is at (0, 0) and its direction is 0. If I click

, where will the sprite end up?
    A. At point X=0, Y=100
    B. At point X=100, Y=0
    C. At point X=100, Y=100
    D. At point X=0, Y=-100

13. Your sprite is presently at (100, 150). What command will you use to move it North by 100 steps?

A.

B.

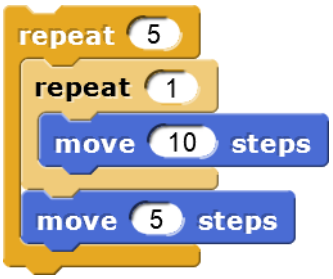
C.

D.

14. How many times will the "turn" command run?

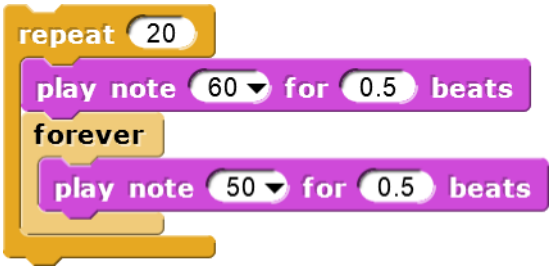
A. 1
B. 2
C. 0
D. 15

15. How much distance will the sprite move after running this script?



A. 35 steps
B. 55 steps
C. 75 steps
D. 15 steps

16. How many times will you hear the note 60 when you run this script?

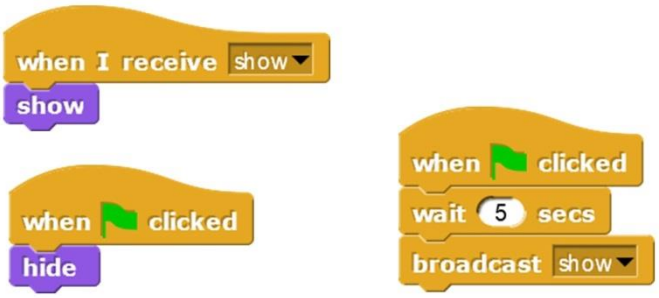


A. 20
B. Infinite
C. 1
D. 0

17. The following two scripts belong to the same sprite. What will happen when you click the green flag?

A. Nothing will happen, because a sprite cannot send a broadcast to itself.
B. The sprite will grow by 10%.
C. The sprite will grow continuously until it fills the screen.
D. The sprite will shrink by 10%.

18. See the two sets of scripts for two separate sprites.



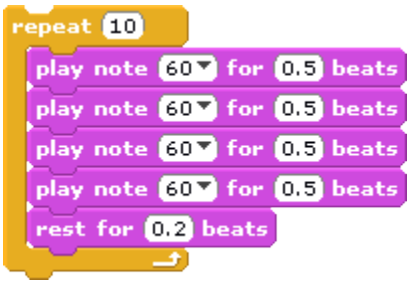
Sprite 1 **Sprite 2**

When you click the Green Flag, what will happen?

A. Sprite 1 will hide forever.
B. Sprite 2 will hide after sprite 1 becomes visible.
C. Sprite 1 will hide after 5 seconds.
D. Sprite 1 will become visible after 5 seconds.

19. In a Snap program, one sprite can talk with another sprite by:
    A. Using the "Say" command
    B. Using the "Send message" command
    C. Using the "broadcast" command
    D. Using the "ask" command

Answer questions 20 and 21 in the context of the script below.



20. If we change the number for “beats” to 0.25, will the song play faster or slower?
    a. Faster
    b. Slower

21. If we change the number for “beats” to 1, will the song play faster or slower?
    a. Faster
    b. Slower

22. As you move right, the X-value gets...
    A. Bigger
    B. Smaller
    C. Stays the same

23. As you move left, the X-value gets...
    A. Bigger
    B. Smaller
    C. Stays the same

24. As you move up, the Y-value gets...
    A. Bigger
    B. Smaller
    C. Stays the same

25. As you move down, the Y-value gets...
    A. Bigger
    B. Smaller
    C. Stays the same

26. If you run the

command, the new position of the sprite on the screen depends on the current position.
    A. True
    B. False

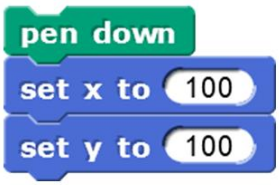
27. If you run the

command, the new position of the sprite on the screen depends on the current position.
    A. True
    B. False

28. If you run the

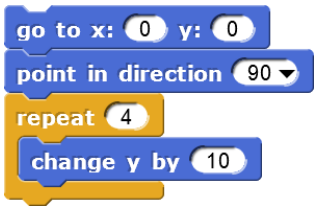
command, the new position of the sprite on the screen depends on the current position.
    A. True
    B. False

29. Will the following two scripts do the same thing? Assume that initially the sprite is at the center (x=0, y=0).



A. Yes
B. No

30. What would the Y position of the sprite be after running the following script?



A. 40
B. 10
C. 0
D. 14

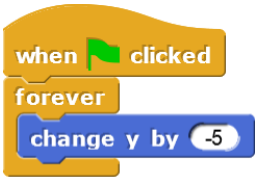
31. If we give the command

, where would the sprite be positioned on the screen?
A. Top right
B. Top left
C. Bottom right
D. Bottom left

32. What happens when a sprite runs the command

?
A. The sprite moves smoothly by 100 steps
B. The sprite jumps by 100 steps
C. The sprite jumps to a point that is 100 steps away from the center of the screen
D. Nothing happens because the sprite doesn't know where to move

33. What will happen when we click on the Green flag?

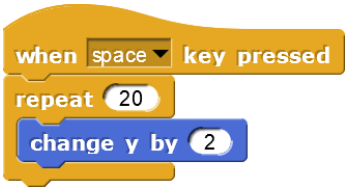


A. The sprite will move down 5 steps once, and then stop.
B. The sprite will move down 5 steps at a time continuously.
C. The sprite will move up 5 steps at a time continuously.
D. The sprite will move 5 steps down until it reaches the bottom of the stage and then move up.

34. The following script makes the sprite fall down as if pulled by gravity. Which number will make it fall faster?

A. -0.5
B. 0
C. 2
D. -2

35. The following script makes the sprite jump upwards. How can we make it jump higher?



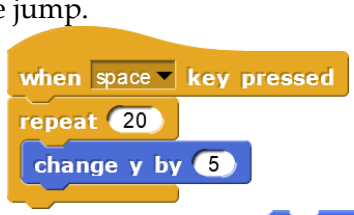
A. By repeating more number of times.
B. By using the IF command.
C. By repeating less number of times.
D. By changing y by a smaller number

36. How much distance will the sprite move according to the following script?

A. 150
B. 250
C. 350
D. 750

Answer questions 37, 38, and 39 in the context of the following script:

This script makes the sprite jump.



37. What will happen if we change the input of to 10?

    A. The sprite will not jump any more when the SPACE key is pressed.
    B. The sprite will take a shorter jump when the SPACE key is pressed.
    C. The sprite will take a longer jump when the SPACE key is pressed.
    D. The sprite will move down when the SPACE key is pressed.

38. What will happen if we remove the REPEAT block from this script?
    A. The sprite will not jump any more when the SPACE key is pressed.
    B. The sprite will take a shorter jump when the SPACE key is pressed.
    C. The sprite will take a longer jump when the SPACE key is pressed.
    D. The sprite will move down when the SPACE key is pressed.

39. What is the size of the jump in the above script?
    A. 100
    B. 20
    C. 5
    D. 50

Programming problems:

(Refer to Appendix A for links to demo videos of these problems.)

Vibration:
Draw two vertical walls and make a ball bounce between them.

Car:
A girl drives her car continually left to right. The car disappears at the right edge and reappears at the left edge.

Knock knock joke-1:
One sprite tells another a "knock, knock" joke. Use "wait" commands for synchronization.

Basketball dunk-1:
Load the “Hannah” and “basketball” sprites from the Snap library, and draw a basketball hoop. Using Hannah’s 3 costumes create a program in which Hannah dribbles the basketball, runs to the hoop and dunks the ball in the basket.

Grow:
Write a program in which one sprite acts as a button and when clicked, it increases the size of another sprite.

Knock knock joke-2:
Modify the program in which one sprite tells another a "knock, knock" joke. Use broadcasting for synchronization.

Bat and dragon:
There are two sprites: Bat and Dragon. Dragon is facing away. Bat flies to Dragon when the space key is pressed. Bat says "BOO!" when it stops. Dragon then turns around and says, "You scared me!" Bat turns around and flies away.

Security camera:

This security camera shows different rooms in each of which something different is happening.

Basketball dunk-2:

Modify the program above to use broadcasting to synchronize the player and the ball.

Me-time:

A girl is dancing when her friend enters and asks her if she wants to play. She first says "she wants some ME time" but then the letter M spins to become W and it becomes WE time.

Student teacher:

Show a short lesson session between a teacher and a student. Use any topic of your choice.

Notes and guidelines for programming problems:

Vibration:
Consider using a pair of "glide" commands.

Car:
Consider using a pair of "go to x, y" and "glide" commands.

Knock knock joke-1:
This would basically be a simple dialogue with the pauses implemented using the "wait" command.

Basketball dunk-1:
Load the sprites "Hannah" and "basketball". You will need to paint the basketball or import an image from outside. Hannah has 3 costumes that you can use for this animation. The first two do not include a basketball, but the third one does. That is why you will need the "basketball" sprite for the first two costumes.

The animation has 4 parts as listed below. You would need to use a "timeline" to synchronize all actions. For example, in step 1, if Hannah takes 2 seconds to complete her jump, the basketball sprite will need to hide for 2 seconds.

1. Hannah dribbles the ball. Hannah would switch to her dribbling costume. For dribbling, you will need to position the basketball near Hannah's hands and then use the glide commands for the dribbling motion.
2. Hannah holds the ball and moves towards the basket. You can accomplish this once again by positioning the basketball near Hannah's hands and then making them both use the same glide command.

3. Hannah jumps to the hoop. Hannah would switch to her jumping costume in which she has a ball in her hands. Obviously, you will need to hide your ball sprite and then just use a glide to show the jump.
4. Hannah dunks the ball; the ball drops through the hoop down to the ground. Hannah returns to the ground. Hannah would switch to her dribbling costume. Hannah's motion would be a reverse glide. The ball sprite would appear near the hoop and would glide down to the ground.

Grow:

When you click on the button, it sends a broadcast message to the other sprite, which then grows in size.

Knock knock joke-2:

Modify the earlier program by removing all "wait" commands, and using broadcast messages instead. For example, when sprite1 says something and expects sprite2 to respond, sprite1 will send a broadcast message, and sprite2 will respond upon receiving that message.

Bat and dragon:

This program consists of 3 parts:

1. The bat flies towards the dragon. This is a matter of combining motion with costume change.
2. The bat scares the dragon (who is facing away). This involves exchange of a broadcast message.
3. The bat turns around and flies away from the dragon. This is the same as step 1 except in the reverse direction.

Security camera:

We have a security system that shows five different scenes (or cameras). Each scene consists of a background image and one or more sprites. The stage owns the main script, which is a series of "switch background" commands. For every scene (or camera), the script first sends a "hide" broadcast which makes everyone

hide, and then an appropriate broadcast message for the relevant sprites (e.g. the ballerina for the dance-floor) to show up and do their thing.

Basketball dunk-2:

Modify the earlier program by removing all "wait" blocks and using broadcast messages instead. For example, when Hannah is ready to jump to the hoop, she will inform "basketball" via a broadcast message so that it can hide.

Me-time:

The program is entirely controlled through broadcast messages. Initially the girl is doing her dance and the story begins when the boy enters the room. The text messages appearing on the stage would be separate sprites.

Student teacher:

In this program, the stage is an active participant. The text appearing on the blackboard is basically a collection of stage costumes (background images). The dialog between the student and teacher can be purely time-synchronized (i.e. by using wait commands), but after that use broadcasting to synchronize teacher's speech with appropriate background change. You could use "next costume" since there is a well-defined order in which the screens are supposed to appear. Tune the program's pace with additional wait statements if necessary.

Pressing "a" brings up the last background which shows the answers.

Unit 2: Interactive Animation

We should introduce children and young adults to the concepts of computer science through courses designed not to create career programmers (and to filter out the non-programmers), but rather to give them the essential insights and ideas in the field. – Abhay B. Joshi

Concepts practiced in this Unit

- All concepts of previous units
- Events
- Reset script
- Graphic effects
- Concurrency
- User interaction using keyboard
- Conditions: YES/NO questions
- Sensing touch
- Conditionals (IF)
- Conditionals (IF-Else)
- Stopping scripts

Concept Review Questions:

1. Your game program first shows a "help" screen, and you would like to change the screen to the “game” screen when the user clicks anywhere. Which event block would you use to sense this click?

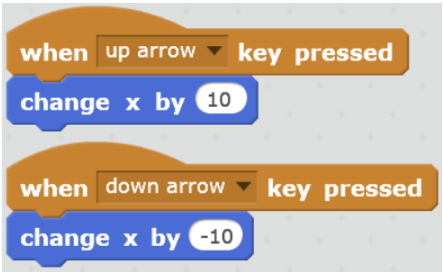
A.

B.

C.

D.

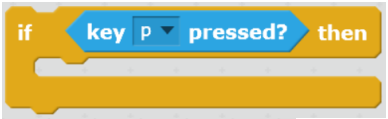
2. What will the following scripts do?



   A. Move the sprite up and down
   B. Move the sprite diagonally
   C. Move the mouse pointer right and left
   D. Move the sprite right and left

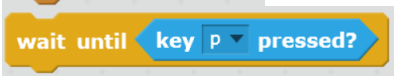
3. Which of the following is not a graphical effect?
   A. Color
   B. Whirl
   C. Glide
   D. Pixelate

4. You want to make your sprite to perform some actions every time the key "P" is pressed. Which block would you use?

A. 

B.

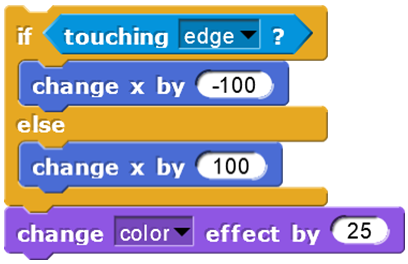
C.

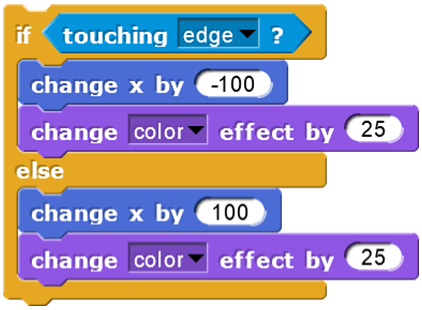
D. 

5. What will we get with the following script?

A. Color will change only once
B. Color will not change at all
C. Color will continuously change 50 times
D. Color will set to black.

6. Will the following two scripts do the same thing?


if touching edge ?
change x by -100
else
change x by 100
change color effect by 25


if touching edge ?
change x by -100
change color effect by 25
else
change x by 100
change color effect by 25

A. Yes
B. No

Programming problems:

(Refer to Appendix A for links to demo videos of these problems.)

Rolling ball-2:
Modify your rolling ball program to make it roll both ways: left to right and right to left. Explain the limitation of this program. Use concurrency (but without broadcasting) to implement this.

Rolling ball-3:
Modify your rolling ball program such that the ball will roll both ways continuously: left to right and then right to left. Use broadcasting to achieve this.

Diagonal 3D:
Load a background that has a feel of depth, e.g. "desert". Make a sprite move diagonally between the lower-left and upper-right corner. The sprite should appear as if it is moving away and closer as it moves (Hint: change size).

Traffic light-1:
Write a program in which a car moves continuously between the left and right edges while obeying a traffic signal. That is, the car moves only on Green and stops on Red. (Use broadcasting for synchronization)

Break the brick:
When the ball collides with the brick, the brick breaks into pieces.

Pet the dog-1:
When the mouse pointer touches your dog sprite, it barks. It barks as long as the pointer is touching.

Ball at the edge:
Have a ball sprite. The reset script makes it go to the center (x=0, y=0). The "up arrow" key should move it up slightly, but not beyond the screen edge. (That is,

if the ball is touching the edge, the ball should not move if the up arrow key is pressed.)

Ball and paddle-2:

Modify your paddle and ball program to make the program stop when the ball touches the ground (the lower edge of the screen).

Notes and guidelines for programming problems:

Rolling ball-2:

The rolling motion is accomplished by splitting the motion into two components: one component is a straight-line motion along the X axis (for which you can use "glide"), and the other component is the spinning around itself. When these two are combined you get the rolling effect. Since you want the ball to roll in both directions, you can use "repeat" for spinning and calibrate the repeat count (i.e. make it match the "glide" time) by trial and error. There are two problems with this approach: (1) The two motions go out of sync after some time. (2) If you change the glide time, the program doesn't work.

Rolling ball-3:

The problem with rolling-ball-2 is that the tuning of concurrent scripts depends on the repeat count, which is at best approximate. So, after a while, you can see that the concurrent motions go out of sync.

The solution is to synchronize the two motions using broadcasting. You will need two sets of scripts since there are two directions. The main script would control the pair of glides and control the spinning scripts by starting them using broadcast messages and stopping them using "stop" commands.

Diagonal 3D:

This program shows 3 things happening concurrently: (1) The girl gliding across the screen diagonally. (2) The girl changing costumes to appear walking. (3) The girl changing size for the 3-D effect. You can achieve this by having 3 concurrent scripts. In addition, the girl reverses her direction, for which you will need to use separate scripts which need to be synchronized using broadcast messages. Stopping scripts can be achieved using the "stop" commands.

Get everything working in one direction first. And then duplicate the scripts and make appropriate changes for the other direction.

Traffic light-1:

You can implement the traffic light using costumes. The car's motion is straightforward. It can be coordinated with the traffic light using broadcast messages. For example, when the traffic light changes its costume to red, it will send a broadcast message to stop its motion script.

Break the brick:

When the ball touches the brick wall, the wall changes its costume to make it appear broken. The "wait until" block can come handy.

Pet the dog-1:

The dog sprite senses "touching pointer" in a forever loop.

Ball at the edge:

Consider using the "IF-Else" block inside your up-arrow event script. Also note that moving 0 steps is the same as "being stopped".

Ball and paddle-2:

Take your earlier ball paddle program, and make the ball sense touching the ground. Use the "stop" command.

Unit 3: First Game Project

Is it possible that software is not like anything else, that it is meant to be discarded: that the whole point is to always see it as a soap bubble? – Alan J. Perlis

Concepts practiced in this Unit

- All concepts of previous units
- User interaction using mouse pointer
- Conditionals (Wait until)
- Variables – numbers
- Variables as sliders
- Keyboard events (polling)
- User input (buttons)

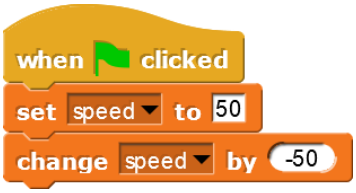
Concept Review Questions:

1. A program provides a "click-button" labeled "Jump". What might the purpose of this button be?
    A. When the user clicks on it, the button will jump.
    B. When a sprite touches the button, the sprite will jump.
    C. When the user clicks on it, some sprite in the program will jump.
    D. It serves as the title of the program.

2. See the script below and select the correct choice.

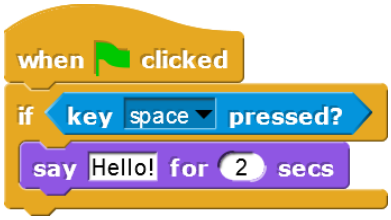
    A. The sprite will say "name" when Green flag is clicked.
    B. The sprite will say "Danny" when Green flag is clicked.
    C. The sprite will not say anything when Green flag is clicked.

3. Your program has a slider variable "speed" with min=5 and max=50. What will happen after running the script below?



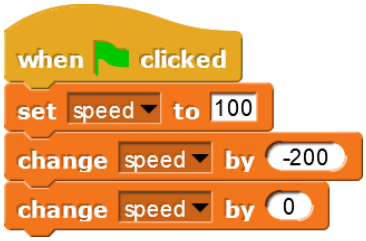
    A. The speed variable will contain 5 because it cannot go below min=5.
    B. The speed variable will contain 50 because it cannot go below min=5.
    C. The speed variable will contain an unknown number.
    D. The speed variable will contain 0.

4. See the script below and select the correct answer.



   A. The sprite will say "Hello!" if the user presses the SPACE key.
   B. The sprite will not say anything if the user presses the SPACE key.
   C. The sprite will say "Hello!" if the user clicks the Green flag.
   D. The script is incorrect because it requires the event "When SPACE key pressed".

5. What will "speed" contain after running the following script?



   A. 0
   B. -100
   C. -200
   D. 300

6. What is the value of x at the end of this script?

A. 2
B. 7
C. 5
D. 11

7. Let's say we have a variable called "score" which contains some number N. Assume that the

block did not exist. How would you achieve the effect of changing the value of “score” by 50?

Answer questions 8 and 9 using the following script which plays notes from 61 to 70.

8. In order to play notes 70 to 61 (in reverse order), how should the command inputs be changed? Write the inputs separated by commas. For example, the input sequence for the script above is 61,10,0.5,1.

9. What would the input sequence be if we wanted to play only odd numbered notes from 61 to 70?

Programming problems:

(Refer to Appendix A for links to demo videos of these problems.)

Earth and sun:
Load "earth" and "sun" sprites. When green flag is clicked earth should continuously rotate around the sun. Next, add a slider variable "speed" using which you should be able to control the speed of earth's rotation, without changing its orbit and distance from the sun.

Slanted maze using keys-1:
If the maze in a pacman program consists of only horizontal and vertical pathways you can move the pacman using the keyboard (e.g. arrow keys). What if the maze also consists of some pathways that are straight but slanted? Devise a way to still only use the keyboard to move the pacman. (Hint: You could assign keys to turn the pacman.)

Beware of bugs:
This is a simple game in which the screen shows an assortment of food (e.g. apples) and bugs. Your sprite (e.g. shark) gets points for eating food but loses points upon touching bugs. The game starts when you press SPACE after which the main sprite follows the mouse pointer.

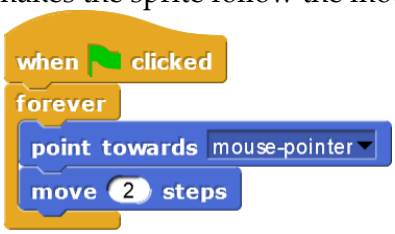
Planet info tool:
Create a solar system teaching aid in which an information bubble pops up when you take your pointer on top of any of the planets. For instance when you move the pointer over Jupiter, the bubble might say "Jupiter: the largest planet with 20+ moons." (Note: No clicking should be required.)

User interaction practice:
Q1: We want to say “Hello” when the SPACE BAR is pressed. But this should happen only once. The second time you press SPACE BAR, nothing should happen. Write the script.

Q2: We want to say “Hello” every time the SPACE BAR is pressed. Write the script.

Q3: We want to move the sprite continuously until the SPACE BAR is pressed. Write the script.

Q4: The following script makes the sprite follow the mouse pointer continuously:



The problem is the sprite starts flickering when it reaches the mouse pointer. Can you fix this problem such that the sprite just stops moving after reaching the mouse pointer?

Q5: The sprite should not follow the pointer, but it should jump to where the pointer is every time it is clicked.

Notes and guidelines for programming problems:

Earth and sun:

Circular motion is achieved when you move and turn continuously. Once you tune your script to get a suitable circular motion, think of the relationship between the move and turn and the circumference of the circle. Then you can use a variable for one of them (say the input of "move") and find an expression for the other (the input of "turn"). Here is the relationship:

Circumference = input of move * (360/input of turn)

Slanted maze using keys-1:

We know how to use arrow keys to move up/down and left/right. In a maze that contains slanted pathways, this won't work. One idea is to use right/left arrow keys for forward/backward motion and use up/down arrow keys to change sprite's orientation.

Planet info tool:

The script would be virtually identical for all planets, except for the pop-up message. We want each planet to bulge slightly and pop-up its description whenever the mouse pointer hovers over it. As soon as the pointer leaves the planet, it should shrink back and remove the pop-up. The script would have a forever loop inside which there would be two conditions to look for: (1) when the planet is touching the mouse pointer, and (2) when the planet is NOT touching the mouse pointer. The problem with this is, the planet will continuously grow or shrink. We need some way to grow/shrink just once after the pointer has moved over or away. A variable can do this trick.

User interaction practice:

Q1: Hint: The usual "when key pressed" won't work in this case. Q3: Consider using conditional looping. Q4: Consider using the NOT Boolean operator. Q5: You could either use "when stage clicked" event or the "mouse down" condition.

Unit 4: Advanced Game

Intellectual activity does not progress by going step-by-step from one clearly stated and well-confirmed truth to the next. On the contrary, the constant need for course corrections, which I call "debugging", is the essence of intellectual activity. – Seymour Papert

Concepts practiced in this Unit:

- All concepts of previous units
- Relational operators (=, <, >)
- Variables as remote control
- Built-in variables - properties
- Motion - direction and bouncing

Concept Review Questions:

1. What will the following command do?

   A. The sprite will move to a new location on the screen.
   B. The sprite will move to the mouse pointer.
   C. The sprite will remain where it is.
   D. The sprite's "direction" property will change.

2. Every sprite has a property called "costume #". Let's say you have a sprite with 5 costumes and it is currently using the first costume. So, "costume #" will be 1. Which command can you use to change this value to 5?

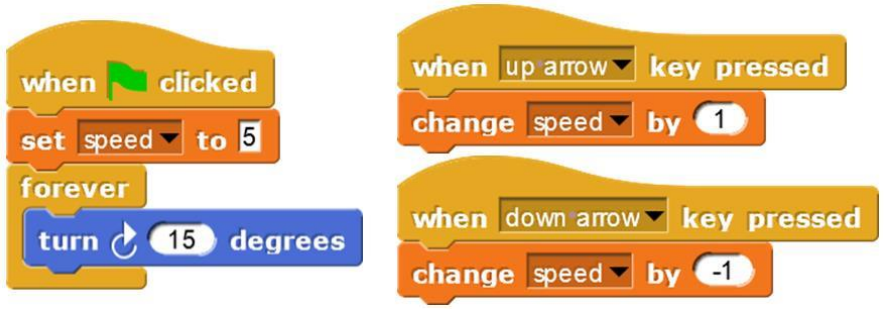
   A.

   B.

   C.

   D.

3. The following scripts are expected to allow us to control the speed of a wheel using arrow keys. But, when arrow keys are pressed, the speed does not change. What is wrong?



   A. The use of "change" command is incorrect.
   B. The turn command needs to use the speed variable as its input.
   C. We must use the "key pressed" conditional instead of events.
   D. Once "speed" is set to 5, it cannot be modified.

4. Is it possible for one sprite to remotely control another sprite's color?
   A. Yes, by using a variable as input to the "set pen color" command.
   B. No, a sprite's color is its own property; no other sprite can change it.
   C. Yes, by using a variable as input to the "set color effect" command.
   D. Yes, by using the STAMP command.

Programming problems:

(Refer to Appendix A for links to demo videos of these problems.)

Rolling ball-4:
Modify your rolling ball program in which the ball rolls both ways continuously: left to right and then right to left. Use variables for synchronization.

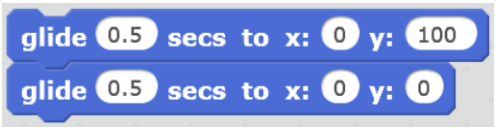
Largest of 3:
The program takes 3 numbers from the user and tells which one is the largest.

Traffic light-2:
Modify the "traffic light-1" program to use variables instead of broadcasting for synchronization.

Weather God:
There are 3 sprites: Johnny, Weather, and Weather-god. The “weather” sprite has costumes to show 3 kinds of weather: sunny, rainy, and unknown. The “Johnny” sprite has costumes to show appropriate response to the weather: happy (for sunny), sad (for rainy), and confused (for unknown). When you click the Green flag, the Weather-god appears and asks what kind of weather you want. Depending on what you type the appropriate weather should show up and Johnny should make the appropriate kind of face. (Note: This program requires the ASK command.)

Bouncing under gravity:
Show a ball bouncing between the left and right edges of the screen but also obeying gravity, i.e. it will also bounce off the ground like in real life causing a curvy motion.

Jump anywhere:
To make the cat jump you can use a pair of glide commands as shown below. Modify this idea (using “properties”) such that the cat will jump anywhere you place it on the screen. (As shown in the video, create multiple platforms. User can move the cat to any platform. When clicked, the cat makes a few jumps.)



Free motion:

This program is based on the use of costumes, but also requires use of the "direction" property. Load the "butterfly" sprite. Make it fly around the screen freely in a realistic manner, i.e. it should not become upside down, nor should it fly facing just straight left-right.

Cat and mouse:

The cat travels around the screen (not chasing the mouse), and whenever it comes too close the mouse runs away. Use ideas in "freemotion" program to move the cat realistically.

Ball and paddle-4:

Modify the ball and paddle program as follows:

a. Make the paddle move horizontally by following the mouse pointer (instead of arrow keys).
b. In version 3, the ball bounces off the paddle at some fixed angle. Change this to make it more realistic; include a small random fluctuation to prevent the ball from getting into a fixed pattern.

Force field:

Imagine that there is a force field (like in Star Trek) which the free-moving sprite cannot cross. It is invisible. Whenever the sprite touches the force field, it gets a shock and is thrown back. The field "shimmers" momentarily when touched.

Train:

This is a train of animals. The engine moves around the screen freely turning randomly every few seconds. The bogies follow in a line keeping a respectful gap in between. The speed slider allows you to control the train's speed. Optionally you can have a "gap" slider that controls the gap between each pair of bogies.

Slanted maze using keys-2:

In the "slanted maze" program written earlier, add this useful feature: pressing the SPACE key should momentarily bring up a dashed line that indicates the current direction of the pacman.

Scrolling with objects:

Create a scrolling landscape background by moving background objects. Load a sprite that has walking costumes (e.g. Avery). The background moves only when the walking sprite walks (using arrow keys). After disappearing at one edge background objects would reappear at the opposite edge.
Create the following versions:

1. All background objects are at the same depth.
2. Make background objects appear in multiple layers, i.e. at different distances from the walker to create the impression of depth.
3. Make the image roll-over smooth, i.e. a background object should disappear smoothly (and not abruptly) at one edge and should reappear at the other end smoothly.

Notes and guidelines for programming problems:

Rolling ball-4:

Instead of sending a broadcast message (as in Version 3), the glide script will set a variable to indicate the direction of motion. The other script will decide the direction of rotation based on the value of this variable.

Largest of 3:

First write an algorithm for this problem and convert it to a Snap script.

Here is one possible algorithm:

```
Given: numbers P, Q, R, and a temporary variable Temp
If P > Q
      Temp = P
Else
      Temp = Q
Endif
If Temp > R
      Declare Temp as the largest number
Else
      Declare R as the largest number
Endif
```

Weather God:

This program involves using a nested IF-Else block. Here is a possible outline:

```
If user says sunny
      Show sunny
Else
      If user says rainy
            Show rainy
      Else
            Show unknown
      Endif
Endif
```

Bouncing under gravity:

This is an interesting challenge of combining 3 things: the ball moving horizontally, it spinning around itself, and also falling down (to obey gravity). There may be other approaches to accomplish this, but a simple way is to turn and move concurrently (i.e. in 2 separate scripts running in parallel). So, if the ball is moving left to right and also turning clockwise, it would curve down to the ground and "if on edge bounce" would make it curve upwards in a real-life fashion. The remaining challenge is to make it do a similar thing while moving right to left. This involves checking the direction of the ball (using "direction" property) and turning clockwise or anti-clockwise accordingly.

Jump anywhere:

Glide commands use x and y coordinates, so, when you use them with fixed values, the jump would always occur between the same two points. To make it work anywhere on screen, we need two things: (1) A way to pick the current x value and use it in both glides: this is achieved by using the property "x position", and (2) A way to pick the current y value and use that as the starting value for the jump: this is achieved by using the property "y position".

Free motion:

The Snap feature to set rotation style "only face left/right" allows you to ensure the sprite does not become upside down when it moves right to left. But the problem is that it then stops turning in any direction other than east or west. For instance, if the sprite is moving in the south-west direction, its orientation remains "west".

On the other hand, setting rotation style to "can rotate" allows the sprite face any direction, but it becomes upside down if it is facing say south-west or north-west.

In this program, we want to fix this problem, such that the sprite actually faces the direction in which it is moving without becoming upside down.

We use the Snap feature to set the rotation style to "can rotate" to ensure the butterfly can turn all around. To eliminate the problem of becoming upside down, we use additional costumes. So, we use the normal set of costumes for all directions facing east (i.e. direction > 0), and their mirror images (flipped vertically) for all directions facing west (i.e. direction < 0). The Snap paint editor has a nifty feature to create mirror images of costumes.

Cat and mouse:

Refer to the "free motion" program to understand how to move the cat around in a realistic-looking way.

The remaining challenge is to keep the mouse at a "safe distance" from the cat. The property "distance to" can be monitored continuously for this purpose. As soon as this distance becomes less than some number (say 100), the mouse should jump away.

Ball and paddle-4:

To track the mouse pointer, consider using the property "mouse x".

To fix the "bounce angle" we will use the ball's "direction" property. If the ball's direction before bouncing off the paddle (either its top or bottom) is "A", it is "180-A" after bouncing. To this, add a small random number such that there is some variation.

Force field:

The force-field essentially creates an area beyond which the sprite cannot go. We could create a rectangular area by using some min and max values of X and Y. For instance, we could say that the "restricted area" is from -175 at the left to 175 at the right, and from 125 at the top to -125 at the bottom. If the sprite happens to go beyond this area, the force field should "shimmer" (like a charged field) and push the sprite away.

Consider using the properties "x position" and "y position" to continuously monitor the sprite's position. Whenever either of them crosses one of the above limits, you could make the background do the "shimmering" and make the sprite jump back into the safe zone (by reversing its direction). See a partial algorithm below:

```
Variables:
maxX, minX, maxY, minY
Sprite moves freely within these coordinates.

Algorithm:
Forever
  wait until x < minX
  turn around
  send message (to signal that you hit the force field)
  wait until x > minX (to ensure the sprite gets out)
End-forever
```

There would be similar scripts for maxX, minY, and maxY.

Train:

The basic idea in a train is this: If one sprite (sprite2) wants to follow another (sprite1), sprite2 needs to (a) point in the same direction as sprite1 and (b) maintain some fixed gap all the time.

The "point towards" block takes care of part (a). Part (b) can be achieved by monitoring the distance by using the property "distance to". Sprite2 should move only if the distance is greater than the expected gap.

In a train, the engine is free to move wherever it wants, the first carriage (carriage1) follows the engine (as described above), the second carriage (carriage2) follows carriage1, and so on.

Slanted maze using keys-2:

The "dashed line" would be an additional sprite that remains hidden normally, but shows up when SPACE key is pressed. Before showing up, it jumps to where pacman is, and aligns itself along the direction of pacman (by using the block "point towards").

Scrolling with objects:

In Snap there is no straightforward way to scroll the background. But, we can create the illusion by scrolling background objects: we use sprites that together represent the background, and move these sprites left or right. Essentially we pick multiple sprites that are supposed to be part of the background and move them left or right as the foreground sprite moves.

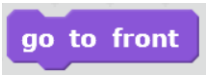
In this program, we will design a city background with tall buildings of various types. Have a foreground sprite (e.g. a walking sprite) that “appears” to walk left or right when you press the left/right arrow keys. The walking sprite really doesn’t move, but the background scrolls left or right. When a building disappears at the left edge, it re-appears on the right edge and vice versa.

“Avery” is the foreground sprite who "appears" to walk left or right when arrow keys are pressed. But in reality, she only walks "in place" without moving at all. That is, she only changes costumes. Instead, the building sprites move right or left (opposite of Avery's direction of motion). When a background sprite hits the left or right screen edge, it reappears at the opposite edge.

Version 2:

To make the moving landscape realistic (i.e. having depth), we need to place objects at different distances. Objects that are farthest will scroll the slowest, and those that are the closest will scroll the fastest.

One tricky matter is how to layer the objects from front to the bottom. Snap offers two layering commands:



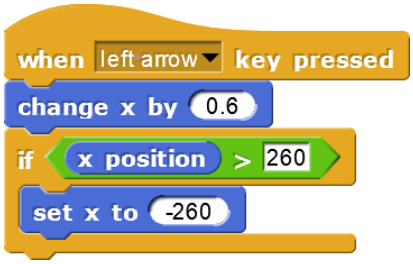
You might think simply having everyone “go to front” first and then asking them to “go back” a number of layers would do it. But, that would have worked if “go to front” took them to the “front” row (at layer 0) to begin with. That is not the case. In fact “go to front” itself creates a multi-layer situation. Whoever executes “go to front” first ends up at the bottom and whoever runs “go to front” last is actually in the front. As a result “go back N layers” may not work as expected because you don’t even know where each sprite is in the layering scheme.

So, the solution to this problem is to have all sprites run “go to front” in the expected order: the one who is supposed to be at the bottom will run it first, and so on. This can be achieved through a series of broadcast messages sent by a coordinator (e.g. stage).

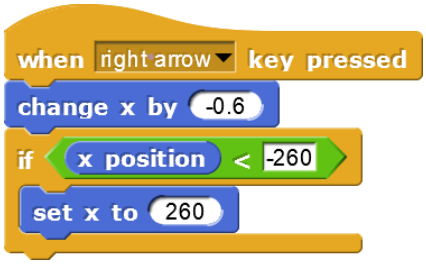
Version 3:
In this version we attempt to fix a problem with “automatic rollover” – i.e. smoothly disappearing at one edge and reappear at the other, and also make it easy to add more background objects.

In Snap the horizontal dimensions of the screen span from -240 to 240. When you continue to change a sprite's X coordinate beyond these values, the sprite gradually moves out of sight. Exactly when a sprite would become completely invisible depends on its width. So, if you want to arrange things such that a sprite moves completely out of sight before reappearing on the other side, you need to adjust the extreme values of X according to the sprite's width.

For example, the following scripts are "tuned" for a sprite that is about 40 pixels wide.



When the X value becomes greater than 260, the sprite becomes invisible. So, it's ok to move it to the other side of the screen.



When the X value becomes smaller than -260, the sprite becomes invisible. So, it's ok to move it to the other side of the screen.

Unit 5: Power of Variables

A great discovery solves a great problem but there is a grain of discovery in the solution of any problem. – G. Polya

Concepts practiced in this Unit

- All concepts of previous units
- Arithmetic operators (+, -, x, /) and expressions
- User input (ASK)
- String variables
- String operations (join, split, letter, length of)
- Variables – as counters
- Random numbers
- Algorithms
- STAMP - creating images
- Conditional looping (repeat until)

Concept Review Questions:

1. Assume that variables "age" and "name" contain the age and name of the user. How many minimum "join" operators will you need to say the following? (Note: Peter and 50 are only examples; the actual values of age and name could be anything.)

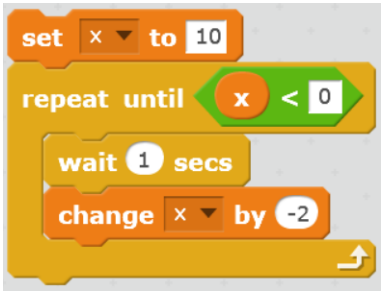
```
The age of Peter is 50 years!!
```

   A. 4
   B. 7
   C. 3
   D. 1

2. Write the script to get a random even number in the range 0 to 50?

3. We want our sprite to point in a random direction, but the angle must be a multiple of 5. How do we achieve that?

4. How long will the following script run?



   A. 6 seconds
   B. 5 seconds
   C. 10 seconds
   D. 11 seconds

5. What will happen when you run the following program on Sprite1:

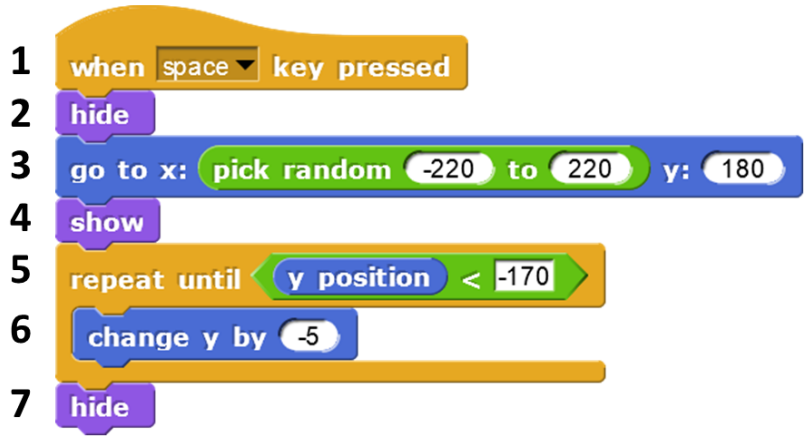
   A. Sprite1 will start changing its costume after touching Sprite2
   B. Sprite1 will touch Sprite2 just before changing costume
   C. Sprite2 will change its costume after waiting for 2 seconds
   D. Sprite1 will stop changing its costume after touching Sprite2

6. Which programming construct can you identify in this algorithm for making a simple fruit salad?

   Peel a banana
   Chop it in small pieces
   Put all the banana pieces inside a bowl
   Peel an apple
   Chop it in small pieces
   Put all the apple pieces inside the bowl

   A. Decision
   B. Sequence
   C. Iteration (loop)
   D. None of the above

Answer questions 7, 8, and 9 based on the script below. Every time you hit the space bar, the sprite goes to a random spot at the top of the screen and falls down until it hits the bottom of the screen.



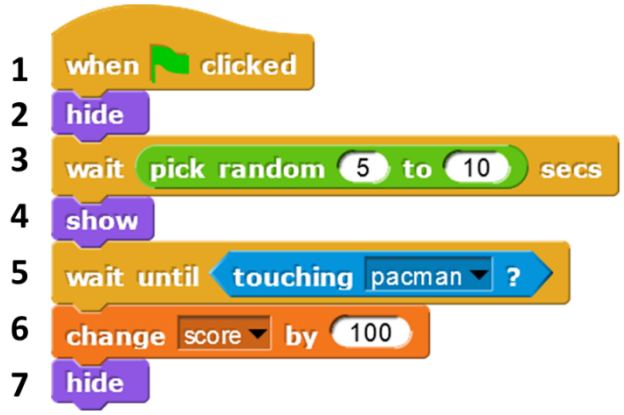
Now, we want to make the sprite start at the bottom of the stage and fly up to the top!

7. What line of the script would you change so that the sprite starts at the bottom of the stage? Type in the number of the line only.

8. What line of the script would you change so that any motion the sprite makes is up instead of down? Type in the number of the line only.

9. We need to change one more line to make sure that the script works the same for flying sprites as it did for falling sprites. What is the last line that we should change?

Answer questions 10 thru 13 based on the script below. The script is written for a "prize" sprite in a pacman game. The prize sprite appears after some time and disappears again when the pacman touches it.



10. What block would you change if you wanted the prize sprite to appear after exactly 15 seconds? (state the line number)

11. Why is it important that block 2 is a "hide" block?
    A. The prize needs to move before it can be visible
    B. The prize has already been collected, so it should disappear
    C. The location of the prize remains secret before it's supposed to be collected
    D. This block isn't actually necessary

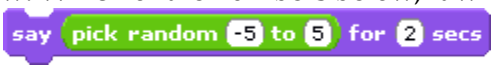
12. Why is it important that block 4 is a "show" block?
    A. Once the prize has been collected, its location should be revealed
    B. The prize changes the stage to look like a theater
    C. The prize needs to appear eventually so that it can be collected
    D. This block isn't actually necessary

13. Why is it important that block 7 is a "hide" block?
    A. The prize provides invisibility to the pacman
    B. The prize has already been collected, so it should disappear
    C. The prize should only be possible to collect for a short time
    D. This block isn't actually necessary

14. See the script below. Which of the numbers below, it will never say?

A. 5
B. 10
C. 8
D. 4

15. See the script below. Which of the numbers below, it will never say?



A. 6
B. 0
C. -3
D. 3

16. See the script below. Which of the numbers below, it will never say?

A. 7
B. 20
C. 24
D. 10

Programming problems:

(Refer to Appendix A for links to demo videos of these problems.)

Fireworks:
A tiny ball shoots up and explodes in a fiery display of colorful explosion. Show each explosion using costumes and graphic effects.

Count letters:
Count the number of occurrences of the given letter in the given string.

Reverse string:
The program asks for a word (string of letters) and reverses it. For example, if you give "programming" the program will show "gnimmargorp". Write another script that will test the correctness of the "reverse" script by supplying known strings.

Is Square:
Examine if the given positive integer is a perfect square.

Simplify radical:
Given a positive radical (number inside square root), simplify it. For example: sqrt(125) in simplified form would be 5*sqrt(5).

Star matrix:
Load the "star" sprite. Using the STAMP command create an 8x8 matrix of stars.

Chessboard:
Using a single square sprite with 2 costumes – one light and one dark – draw the 8x8 chessboard pattern. Use the STAMP command.

Matrix-1:
Using STAMP create an NxN matrix of squares. Ask user for the value of N. Calculate the size of an individual square cell such that the total matrix nicely fits

the Snap screen. For example, if N=1, there would be just one large square that occupies the entire screen. If N=2, the square cell would be half the size, and so on.

Placement of cards-1:

Load any 8 images (costumes or sprites). When SPACE is pressed all these images assemble (on top of each other) in one corner of the screen. When green flag is clicked, these images get laid out one by one in neat 2 rows of 4 images each.

Birthplace of words:

The program shows an icon labeled as “Birthplace”. When green flag is clicked a word pops up repeatedly on the screen – at a random location, with a random size and with a random color. Every time you click on “Birthplace” the word changes. Of course, after a while, the words are recycled. A “clear” button allows you to clear the screen at any time.

Butterfly-1:

A "real" butterfly flies around the screen – its speed and direction change randomly. It also appears to fly away from or closer to the camera.

Show number-1:

Ask the user for a decimal number and display it on the screen (without using the SAY command). Use the stamp command. For example, if the user enters 1234, the program will write "1234" on the screen near the mouse pointer.

Touch 5 times:

Write a script in which the sprite moves around the screen but stops after touching the screen edges 5 times.

Eat donuts:

Green flag creates a few donuts (shown by a variable) which are hidden. When SPACE is pressed they appear one after the other and the shark jumps on and eats each until there are no more donuts.

Guess number-1:

Computer picks a number between 1 and N and user tries to guess it. The player decides N – the upper limit of the range of possible numbers from which the secret number is chosen. Keep track of how many guesses the player has made and declare at the end. Help the player by telling him or her whether the guessed number is too high or too low.

Guess number-2:

Same as above, except the player picks a number between 1 and N and the computer tries to guess it. The player decides N. The player helps the computer by telling it whether the guessed number is too high or too low. Computer guesses using random guessing. The random guess uses the revised range after every guess.

Guess number-3:

Same as above, except computer guesses using binary search: each time it takes the middle number of the range.

Sweets:

How many combinations of 25c pastries and 20c cookies can I buy in $5? For example, 8 pastries and 15 cookies will add up to $5.

Weights:

Find all combinations for weights of 12, 20, and 30 kg such that their total is 300 kg. For example, 10 weights of 12 kg, 6 weights of 20 kg, and 2 weights of 30 kg add up to 300 kg.

Coins:

How many combinations of pennies, nickels, dimes, and quarters, will give $1? For example, 10 pennies, 9 dimes, 2 nickels, and 1 quarter will add up to $1.

MCQ Quiz:

This program demonstrates how a quiz containing multiple-choice questions can be designed in Snap. Pick any questions of your interest and design this quiz. After clicking the "start quiz" button questions would be presented one at a time, and user would be allowed to enter the answer. Optionally, instant feedback (right or wrong) is provided. At the end of the quiz, the score is declared.

Countdown timer:

Ask the user for a time (less than 60 seconds) and count down to 0. Show the timer both as an analog timer and a digital timer. The digital timer should use 2 number sprites and not just a variable.

Pet the dog-3:

When the mouse pointer touches your dog sprite, it barks. But it barks only occasionally.

Notes and guidelines for programming problems:

Fireworks:

This program is primarily a play of costumes that look like exploded fireworks. Load or draw such costumes – as many as you wish – and make the first costume a small dot. When the rocket shoots upward, this dot is what moves upward some random distance, in some random direction (but within a range of directions, because you want the rockets to only go upward). Then the sprite randomly changes into one of the "firework" costumes and goes into a short loop of graphic effects (color, ghost and size effects) so that it appears like an explosion.

Count letters:

You can achieve the objective by using the string operators. Here is the algorithm:

```
Count = 0
C = character to be counted
I = 1
Length = length of the given string
Repeat Length times
      If letter at location I = C
            Increment Count
      End-if
      Increment I
End-repeat
Declare Count
```

Reverse string:

Reversing a string involves reading one string letter by letter from its end and copying it to a new string. Here is the algorithm:

```
Given: In and Out strings.
Length = length of In
I = Length
Out = ""
Repeat Length times
      Out = Out + In[I] // Attach character at the end
      I = I - 1
End-repeat
Declare Out
```

Is Square:

We use a brute-force method of trying every number from 1 until we exceed half of the given number. For example, if N=30, we take numbers I=1 to 15 and see if square of I equals N. Here is the algorithm:

```
Given: N a positive integer
Output: True or False
If N is even
      R = N/2
Else
      R = (N+1)/2
End-if
I = 1
Repeat R
      If I*I = N
            Output = True
            Stop
      End-if
      I = I + 1
End-repeat
Output = False
```

Simplify radical:

This is a matter of extracting factors of N that are perfect squares until there are no more. The test for factor is simple: verify it divides with no remainder.

Algorithm:

```
M is multiplier
M=1
i=2
Repeat until i > N:
      If square(i) is a factor of N
            M = M*i
            N = N/square(i)
      Else
            i = i + 1
      Endif
End-repeat
// Display result
If M = 1
      Print radical could not be simplified
Else
      If N = 1
            Print square-root(original N) = M
      Else
            Print square-root (original N) = M * f(N)
      Endif
Endif
```

Star matrix:

Let us develop the algorithm for drawing an 8x8 matrix of stars:

We will first show for 4x4 which can then be adapted for 8x8.

Pass 1: Draw a single row of 4 stars:

Let's assume the gap between each pair is W.

```
Stamp
Change x by W
Stamp
Change x by W
```

```
Stamp
Change x by W
Stamp
Change x by W
```

Pass 2: Do you see repetition?

Yes, by adding "change x by W"

```
Repeat 4
      Stamp
      Change x by W
End-repeat
```

Pass 3: Now draw 4 rows by reusing the above code 4 times

```
Draw row (reuse the above code)
Go to start of new row:
      Set x to starting x
      Change y by -W
Draw row
Go to start of new row
Draw row
Go to start of new row
Draw row
Go to start of new row
```

Pass 4: Do you see repetition?

```
Repeat 4
      Draw row
      Go to start of new row
End-repeat
```

The above algorithm will give a 4x4 matrix. You now just need to change the repeat counts to get an 8x8 matrix of stars.

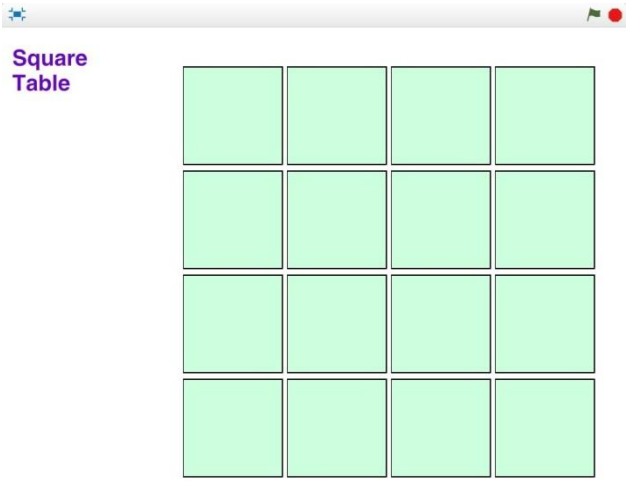
Chessboard:

A chessboard is similar to an 8x8 matrix except it has alternating dark and light squares. We could adapt the algorithm for the "star matrix" program above that draws an 8x8 matrix of stars.

We could use a single "square" sprite that has two costumes: one dark and one light. All we have to do is change costume after every stamp.

Matrix-1:

Several programs require tables with equal number of rows and columns. This program allows you to create a table of size that you specify. For example, if you specify 4, the program will show a table with 4 rows and 4 columns.



Let us now see how we can build this program step by step.

Let's take a look at the main screen of the game and try to point out the different pieces.

Initially, to simplify things a bit, we will just draw a fixed 3x3 table. This could be achieved by stamping a green square sprite 9 times.

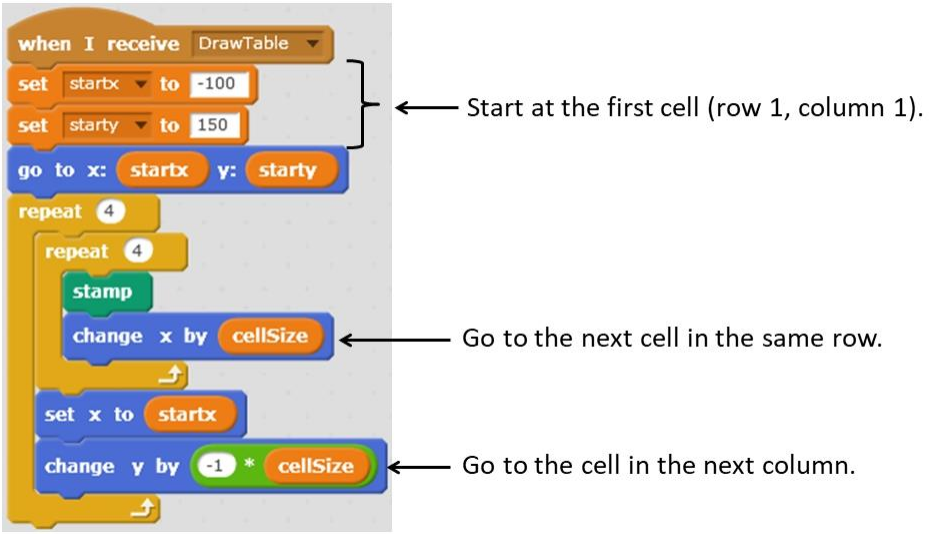
Step # 1
Draw a fixed 3x3 table of filled green squares.

We could use a single filled square sprite and create the table by "stamping" the square sprite one row at a time. The algorithm for this would roughly look like this:

DrawTable Algorithm:

```
Given:
cellSize = width of the square
Go to startx, starty     // top-left corner of the table
Repeat 3
  Repeat 3
    STAMP
    X = X + cellSize     // go to the next cell in the row
  End-repeat
  X = startx             // reset to the beginning
Y = Y - cellSize         // go to the next row
End-repeat
```

Write your own script and then compare it with the one given below.



Step # 2

Draw table of any size.

Deriving R from N is easy, as shown below:

```
R = 1
Repeat until R*R >= N
  R = R + 1
End-repeat
```

The algorithm to draw the table will change only slightly from Version 1. Instead of using 4 as repeat count, we will use R computed above.

DrawTable Algorithm:

```
R = number of rows (same as columns)
cellSize = width of a cell
Go to startx, starty     // top-left corner of the table
Repeat R times
  Repeat R times
    STAMP
    X = X + cellSize     // go to the next cell
```

```
  End-repeat
  X = startx              // reset to the beginning
Y = Y - cellSize
End-repeat
```

Placement of cards-1:

We want to arrange 8 cards in 2 neat rows of 4 cards each. The arrangement essentially looks like a table of 2 rows and 4 columns. We will call each location in this table as a "cell". These cells can be numbered 1 to 8 as shown below:

1	2	3	4
5	6	7	8

To place a card in a cell, we could just use the "Go to x, y" command if we knew the X and Y co-ordinates of the cell (i.e. of its center).

Each picture card (i.e. its respective sprite/clone) can be assigned an id 1 thru 8 based on the sequence of its birth.

We need someone to assign these cells one by one to each picture card in order (i.e. 1, 2, 3…, 8). How do we do that?

The first row consists of 4 cards all of which will have the same Y coordinate. So the following algorithm will work:

```
Set I = 1 (to count the number of cards)
Set Y (Y position of the 1st card)
Set X (X position of the 1st card)
Repeat 4
      Tell picture card I to place itself
      X = X + 100 (assuming 100 is more than the width of
      each card)
      I = I + 1
End-repeat
```

What will each card do?

```
Go to x position X and y position Y
```

For the 2nd row, we just need to change Y:
Y = Y – 140 (assuming 140 is more than the height of each card)

And then, we can repeat lines 3 to 5 of the above algorithm.

Combining all this together in a single algorithm, we get:

```
Set I = 1 (to count the number of cards)
Set Y (Y position of the 1st card)
Repeat 2
      Set X (X position of the 1st card)
      Repeat 4
            Tell picture card I to place itself
            X = X + 100 (assuming 100 is more than the
            width of each card)
            I = I + 1
      End-repeat
Y = Y - 140
End-repeat
```

Birthplace of words:

We could have a main "word" sprite with multiple costumes – each depicting a different word. When green flag is clicked, this sprite would go into an infinite loop, repeatedly stamping itself all over the screen, each time randomly changing size, color, and location. Clicking on the "birthplace" sprite would force the "word" sprite to change its costume and thus print a different word.

Butterfly-1:

We know how to use concurrent scripts to achieve multiple effects in parallel, e.g. in this case motion, size change, costume change, etc. To this, we need to apply some randomness so that the overall effect is closer to a real butterfly.

We could have separate parallel (concurrent) scripts that randomly change the butterfly's speed, direction, and size. To control the speed, we could use a variable. Change the size gradually (using repeat) and ensure there are limits on how large or small the butterfly can grow.

Show number-1:

The main challenge in this program is to convert the given number into something that can be "stamped" on the screen. One way is to have a sprite with 10 costumes – one for each digit 0 to 9. Then, if we can figure out how to split the given number into a sequence of digits, we can use these costumes to create an image of the number on the screen.

For example, let's say the given number is 1476. We read each digit from left: the first one is "1", so we will stamp costume "1". The second digit is "4", so we will stamp costume "4", and so on.

It so happens that it is quite straightforward in Snap to split a number into its individual digits. We can employ string operators for this purpose.

Here is the overall algorithm:

```
W = distance between two digits + width of each costume
S = <given number>
I = 1
Repeat <number of digits in S>
      Digit = I'th letter in S
      Change to costume for Digit
      Stamp
      Change X by W
      Increment I
End-repeat
```

Touch 5 times:

Consider using the "repeat until" block.

Eat donuts:

The number of donuts is picked at random and is stored in a variable. The "donut" uses "repeat until" to display itself and informs the shark through a broadcast. Then it is just a matter of the shark doing some nifty animation of jumping on the donut and eating it.

Guess number-1:

Ask the user to guess the number in a forever loop. If the guess is correct, stop the script. Otherwise, give some guidance (as to whether the guess is high or low) and ask again.

Guess number-2:

This is a converse of version 1 above since the computer guesses the player's number. Here is the algorithm:

```
(initialize variables)
max = <ask user>
attempts = 0
mynumber = pick random min to max

(play game)
guess = <prompt user to guess>
if guess = mynumber
      Declare win, # of attempts
      stop program
if guess > mynumber
      guess is too high
else
      guess is too low
Ask user to guess again ...
```

Guess number-3:

In Version 2 above, the computer guesses randomly – which is not only inefficient, it can potentially never reach the answer!

Using the "binary search" algorithm, which basically means taking the middle of the available range as the next guess, the computer can guess your number with certainty and that too in a small number of attempts. For example, let's say the user's number (42) is within the range 1 to 100. Computer would guess 50 as the answer. The user would respond saying "Too high", which means the new range is 1 to 50. The next guess would be the middle of this range, which is 25. And so on.

Sweets:

Let's take the single pastry problem first:
How many 25c pastries can I buy in $5?

Equation:
25x = 500
x = 20

Now, let's consider two pastries:
How many combinations of 25c pastries and 20c cookies can I buy in $5?

Equation:
25x + 20y = 500
Multiple pairs possible: e.g. (20, 0) (0, 25)

Here is a one-line algorithm to find all possible pairs:

```
For every combination of (x, y) check if it meets the
condition given in the equation.
```

Weights:

You can solve this problem using the same approach as "sweets" above.

Coins:

You can solve this problem using the same approach as "sweets" above.

MCQ Quiz:

Each question is a backdrop. The upper half of the backdrop is the question and the lower half presents the choices. The main script simply sends out a broadcast message for each question. A separate script for each question then presents the question, receives the reply and provides feedback and scoring. Two variables keep track of correct and incorrect answers.

Countdown timer:

If user enters a number >60 use the "mod" operator to bring it down to less than 60. You can use a "repeat until" loop to count down to 0. For every elapsed second the analog pointer will turn anti-clockwise 6 degrees. The digital timer can be implemented using number sprites that have costumes for digits 0 to 9. You cannot use "next costume" because you want to count down. Use "switch costume" command instead. You can use the "costume #" property to switch costumes. Here is the algorithm for the 0'th place digit:

```
For every second
If costume # = 1 (which is digit '0')
      Switch to the last costume (which is digit '9')
      Ask 1st place digit to decrement
Else
      Switch to the previous costume
End-if
```

Pet the dog-3:

Use the random operator to pick from a range, say 1 to 10, and make the dog bark only if the outcome is within a sub-range, say, 1 to 5. That way, the barking will become occasional instead of every time the pointer touches the dog.

Unit 6: Clones

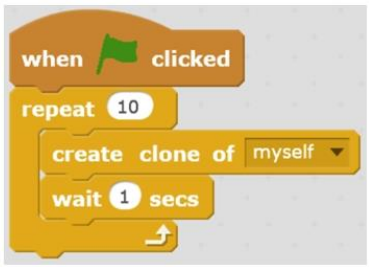
The only way to enjoy programming is to actually do it! – Abhay B. Joshi

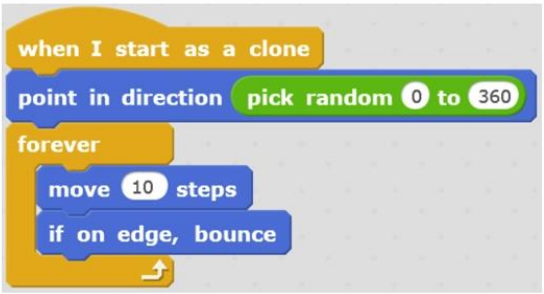
Concepts practiced in this Unit

- All concepts of previous units
- Object oriented programming (OOP)
- Clones
- Motion – piggybacking another object

Concept Review Questions:

See the scripts below (for the "cat" sprite) and answer the following 3 questions:





1. How many total cats will you see on the screen?

2. How many cats will be moving around constantly?

3. How can you make all the clones disappear?

4. Once you create clones, how can you destroy them?

   A. By pressing the red STOP button of Snap.

   B. By calling the

command

   C. By having each clone use the

command.

   D. All of the above.

Programming problems:

(Refer to Appendix A for links to demo videos of these problems.)

Spiralizer:

A ball at the center spins around itself while spitting out identical balls continuously, each of which shoots out in a straight line (with pen down) until it reaches a screen edge. This creates a beautiful rotating spiral.

Green spot illusion:

Pink spots in a circular pattern hide/show by turn. If you stare at the cross-hair at the center, after a while you will see a ghost green spot moving around.

Snowfall:

Write a program in which snowflakes continuously pop up from the sky (from random locations) and float down to the ground and then accumulate as snow. Use clones to create this multiplicity of snowflakes. Use a counter to track the number of "live" flakes at any instant.

Star:

This is a slight variation of spiralizer: A ball at the center spins around itself while shooting out particles (tiny balls) continuously in random directions. Each particle shoots out (with pen down) until touching the screen edge. This creates a beautiful live star. (Challenge: Make the particles come out only at angles that are multiple of 5)

Firework rockets:

Program to shoot rockets of firework. Particles shooting out after explosion are clones.

Version 1: single rocket a time
Version 2: multiple simultaneous rockets

Bouncing balls:
Every click on the background brings up a ball which starts bouncing around. Every ball disappears after a while. Every ball may have a different color and size.

Growing family:
The program begins with a single face. When you click on the background, every face on the stage creates a new baby face which crawls out and tries to find a space for itself. If it touches a screen edge it dies. No two faces may overlap each other.

Bumping balls:
The program begins with a blank screen. When you click on the background a ball appears which starts bouncing around. For subsequent clicks, every ball on the stage creates a new baby ball. If it touches a screen edge while trying to find a space for itself it vanishes. Then it starts bouncing around. All balls bounce off the walls and off each other. No two balls may overlap each other.

Show number-2:
Display the given decimal number on the screen. For example, if you enter 1234, the program will write "1234" on the screen near the mouse pointer. Use the clone feature instead of STAMP.

Matrix-2:
Modify the Matrix-1 program such that each cell is a clone instead of a stamp. Each cell should respond to click events.

Rolling ball-5:
This is a variation of the rolling ball. A soccer ball rolls around the screen freely, i.e. it does both translation and spinning concurrently. (It uses a hidden sprite.)

Go to office:
Create an animation in which a person walks to his car, drives to downtown, gets off when you press SPACE BAR and walks to his office.

Notes and guidelines for programming problems:

Spiralizer:

The parent sprite spins around itself while creating clones continuously. Each clone shoots out until touching the screen edge. This creates a beautiful rotating spiral. (Credit: Computer Coding Projects by Carol Vorderman)

You can make just a few changes to the simpler "star" program to get spiralizer:

- parent turns 5 degrees after creating each clone
- clone doesn't turn at all (remove "point in direction")
- clone doesn't change size (remove "set size")
- make the parent sprite smaller

To make things even more fancy change pen color after each "create clone".

Green spot illusion:

This is an optical illusion program. There are pink spots arranged in a circular pattern that hide/show by turn. If you stare at the cross-hair at the center, after a while you will see a ghost green spot moving around.

Each spot is a clone.

Snowfall:

Use a single snowflake sprite and create clones at regular intervals. Each clone will go to a random spot at the top and fall straight down from there. It will stop (and then delete itself) only after hitting the ground. For the "accumulation" effect, consider using "stamp".

Star:

A star emits rays of light continuously. Consider the following points.

- The # of rays (i.e. lines drawn) is fixed.

- There is no order to how the rays (tiny particles) come out. They come out from random places.
- The particles vanish as soon as they touch an edge.
- Pressing "space" any time clears the screen.

Design:

- Each particle is a clone of the original star but much smaller.
- The star only creates clones (and does nothing else).
- Each clone upon birth, sets a random orientation (but only a multiple of 5) and starts moving and drawing.
- Upon touching an edge, clones delete themselves.

Firework rockets:

This program shoots rockets of firework. Particles shooting out after each explosion are clones.

Single rocket version:

You can shoot one rocket at a time. Click Green flag to shoot a rocket.

The explosion causes particles to shoot out in all directions with the same speed and the same distance. They draw lines of the same color.

Design hints:

- Rocket and particle (ball) are costumes of the same sprite.
- Clones (invisible and in particle costume) are first created and kept ready. When the rocket (parent sprite) finishes its upward travel, all clones jump to that point and become visible. They start their outward journey immediately.
- Particles disappear either when they reach an edge or after a fixed duration is over. Parent sprite keeps a count of live clones; when it reaches 0, it clears the screen.
- Particle speed slows down gradually - this is done by using a geometric series.

Additional features (version 2):

- Rocket shoots from a different spot every time.
- Rocket climbs a different height every time.
- Particles are all of different colors.
- Each particle shoots out a different distance. This is accomplished by picking a different starting speed.
- Line color is different every time.

Multiple rockets version:
The main difference in this version is we cannot use line drawings because there is no easy way to selectively clear pen drawing of only a specific rocket.

So we redesign the explosion to just have particles.

- Since there is no drawing, parent does not need to "clear", and hence no need to keep track of # of live clones.
- We need additional sprites for additional rockets. They will have identical scripts, but will require their own unique broadcast messages and their own x, y global variables.

Bouncing balls:

You could use the "when stage clicked" event to sense mouse click. A new clone is created at that time which starts roaming around the screen and deletes itself after a while.

Growing family:

Algorithm:

```
When I Start as a Clone:
Use a random size
Pick a random direction
Move as long as touching another clone
     If touching edge - die!
```

There are two ways to move each clone:

```
Repeat until not color touching color
      Move

Forever:
      If color touching color
            Move
      Else
            Stop this script
```

Bumping balls:

Algorithm:

```
When I Start as a Clone:
Use a random size
Pick a random direction
Move as long as touching another clone
      If touching edge - die!
Move forever
      If touching another clone - bounce
      If touching edge – bounce
```

Show number-2:

This is a slight modification of the program "show number-1". Instead of stamping the image of each digit, we will create its clone and make it visible.

Matrix-2:

As mentioned this is a slight modification of the program "matrix-1". Instead of stamping the image of the square sprite, we will create its clone and make it visible.

Rolling ball-5:

The challenge in this program to achieve both translation (motion in straight line) and rotation. You can't achieve this using concurrent scripts because the act of spinning affects the direction of translation.

You can use a hidden sprite to solve this problem. The hidden sprite would perform the translation motion and the visible ball would simply track the hidden sprite continuously.

Go to office:

This is a simple animation that has two parts. In the first part the car and its driver move to-and-fro together. You could use the "go to sprite" command in a forever loop to ensure the driver moves together with the car. The driver sprite will also need to change its direction according to the car's direction. Consider using "point in direction" and the property "direction of car".

In the second part (when "space" is pressed), the motion stops, the driver stops following the car and goes to its office.

Unit 7: Procedures and Pen Art

To appreciate programming as an intellectual activity in its own right you must turn to computer programming; you must read and write computer programs -- many of them. – Alan J. Perlis

Concepts practiced in this Unit

- All concepts of previous units
- Pen art
- Procedures
- Procedures with inputs

Concept Review Questions:

1. The following program draws a circle.

If we wanted to change the size of the circle, what should we do?

A. Use the "Change size" command
B. Change the input of the REPEAT command
C. Change the input of the MOVE command
D. Change the input of the TURN command

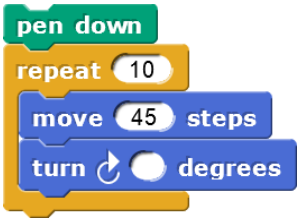
2. If we wanted to demonstrate the beauty of Mozart's music, what might be a good abstraction to use?

A. A painting
B. An audio-clip
C. Words
D. A map
E. A block diagram

3. Which number in the script below needs to be changed to increase the size of the shape?

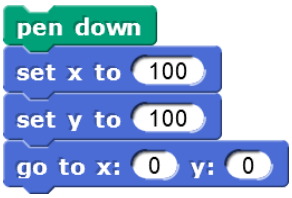
A. 5
B. 50
C. 72

4. We want to draw a 10-sided polygon. What number should be used as input to the Turn block?



A. 10
B. 8
C. 36
D. 20

5. What will the following script draw? Assume that initially the sprite is at the center (x=0, y=0).

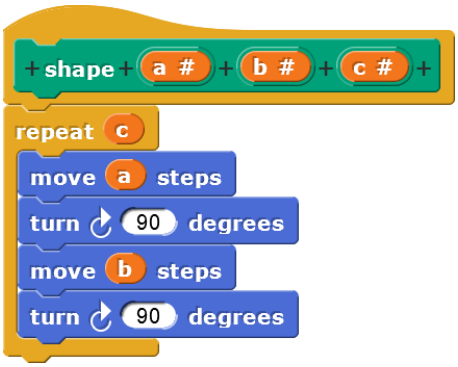


A. Triangle
B. A straight line
C. Two straight lines
D. Nothing

6. The following script draws a circle. What would be its circumference?

A. 360
B. 180
C. 181
D. 720

7. See the following custom block definition.



Assuming that the pen is down, which of the following will draw a rectangle of length 100 and height 50?

A.

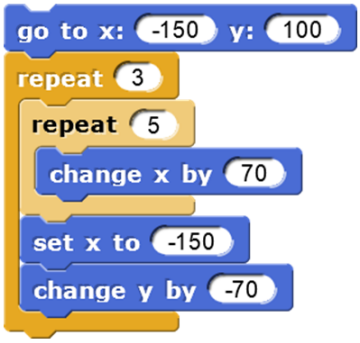
B.

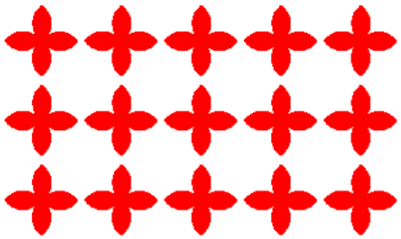
C.

D.

8. We have the following sprite:

We wrote a script to draw a design both of which are shown below:

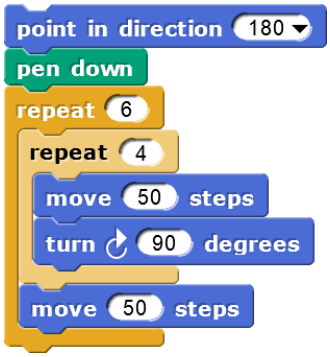


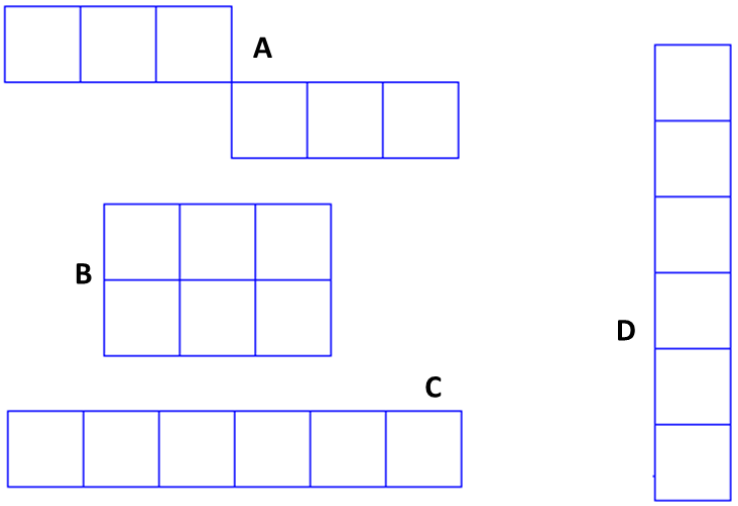


We need to insert the STAMP command in this script to get this design. Where should it be inserted?

A. Inside the first “repeat” block, right before the second “repeat” block.
B. Right after the “go to” block and before the first “repeat” block.
C. Inside the second “repeat” block, right before the "change x" block.
D. Inside the first “repeat” block, right after the "change y" block.

9. What will the script below draw?

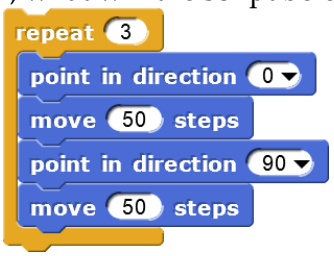


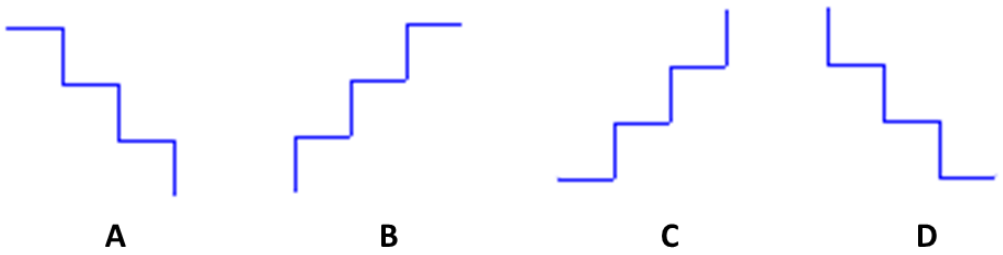


10. The following script draws a square. If we wanted to modify it to draw a hexagon, which instructions need to be modified? Select all that apply.

A. 1 (pen down)
B. 2 (repeat)
C. 3 (move)
D. 4 (turn)

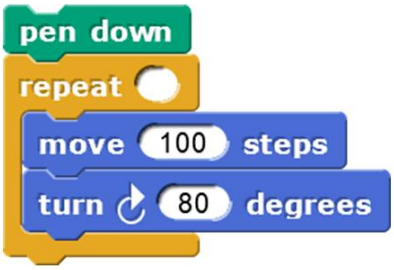
11. Assuming the pen is down, what will the script below draw?

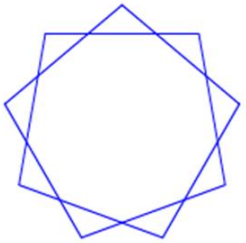




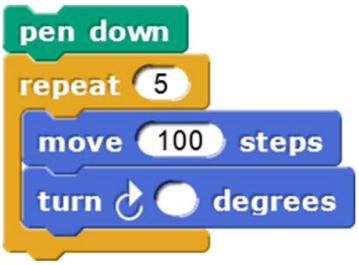
12. The following script draws a hexagon. What would be its perimeter?

13. What input will you give to "repeat" in the script below to get the picture shown?





14. What input will you give to "turn" in the script below to get the picture shown?

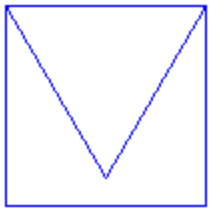


Answer questions 15 thru 18 in the context of the following script.

15. Which part of the script draws the square?
    A. The top repeat
    B. The bottom repeat

16. Which part of the script draws the triangle?
    A. The top repeat
    B. The bottom repeat

17. The script above draws the image below.



Why isn't the triangle on top of the square?

A. Because the blocks inside the second repeat block are in the wrong order
B. Because the sprite is facing the wrong direction after the first repeat block
C. Because the sprite is turning 120 degrees at each corner instead of 60 degrees

18. Which command block in the above script needs to be modified (or replaced) to make the triangle appear on top of the square?
    A. "Move" block inside the second "repeat"
    B. "Move" block inside the first "repeat"
    C. "Turn" block inside the first "repeat"
    D. "Turn" block inside the repeat "repeat"

Answer questions 19 thru 21 in the context of the following passage.

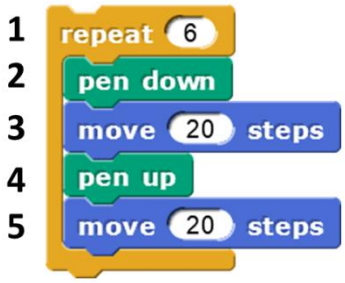
We want to draw the following drawing:

We broke up the problem into three steps.
Step 1:

Step 2:

Step 3:

This is the script for Step 1:



19. What command block should we add to get Step 2?

A.

B.

C.

D.

E.

20. What command block should we add to get Step 3?

A.

B.

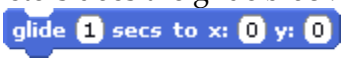
C.

D.

E.

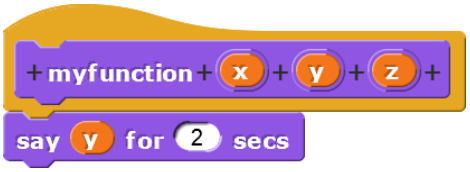
21. Where should we add these command blocks?
    A. Before line 1
    B. Between lines 1 and 2
    C. Between lines 2 and 3
    D. After line 5 and inside the repeat
    E. After line 5 and outside the repeat

22. How many input parameters does the glide block have?



A. 0
B. 1
C. 2
D. 3

23. See the function definition below.

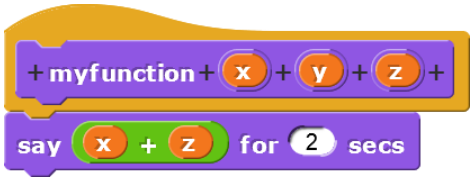


What will the command

do?

A. Say 20
B. Say 30
C. Say 40
D. Say 20 40

24. See the function definition below.

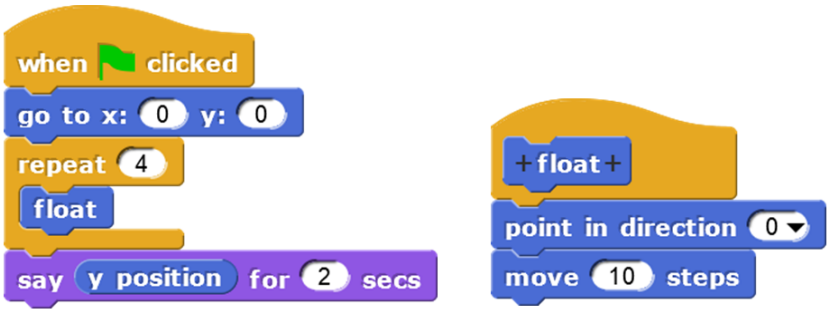


What will the command

do?

A. Say 30+20
B. Say 30+40
C. Say 70
D. Say 50

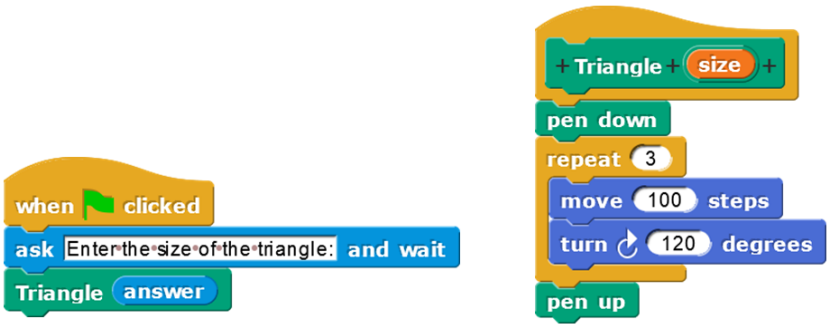
25. We have created the following program for a balloon sprite.



What will the balloon say when we run this program?

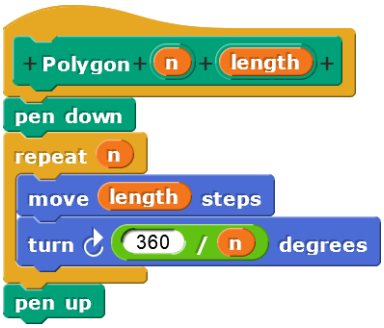
A. 0
B. 40
C. 4
D. 10

26. The following program is intended to draw a triangle of length specified by the user. But, it always draws a triangle of length 100. How can it be fixed?



A. Instead of it should be .
B. Instead of , it should be .
C. Instead of it should be .
D. The function parameter should be named instead of .

Answer questions 27 thru 30 in the context of the following new custom block.



27. What will the command

do?
    A. Draw a square with each side 100
    B. Draw a polygon of 100 sides
    C. Draw a circle of circumference 400
    D. Draw a square with each side 4

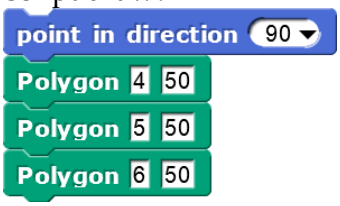
28. What inputs will you give to

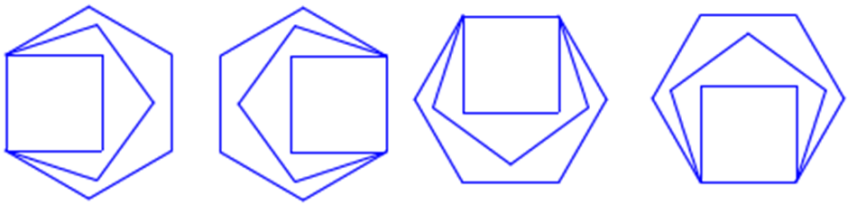
to draw a triangle of length 50? (Use comma to separate the inputs, e.g. 100,100)

29. What inputs will you give to

to draw a hexagon of length 75? (Use comma to separate the inputs, e.g. 100,100).

30. What will the following script draw?





Answer questions 31 thru 33 in the context of the following passage.

We have defined a function called "myfunction" which makes the sprite say numbers.

makes the sprite say numbers 2, 3, 4.

makes the sprite say numbers 5, 6, 7, 8.

makes the sprite say numbers 10.

31. What numbers will the sprite say if you run

?
    A. 4,5,6,7
    B. 3,4,5
    C. 3,4,5,6
    D. 3,4,5,6,7

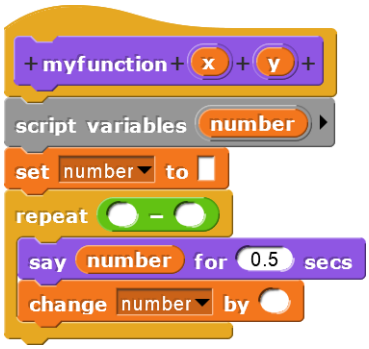
32. What numbers will the sprite say if you run

?
    A. 5,6
    B. 6
    C. 6,6
    D. None of the above

33. What numbers will the sprite say if you run

?
    A. -3,-2,-1,0,1
    B. 1,0,-1,-2,-3
    C. 0,1
    D. None of the above

The new custom block "myfunction" referred to in the above passage, is defined as follows:



Now answer questions 34 thru 36:

34. What would go in the empty cell in line 1?
    A. 1
    B. x
    C. y
    D. y – x

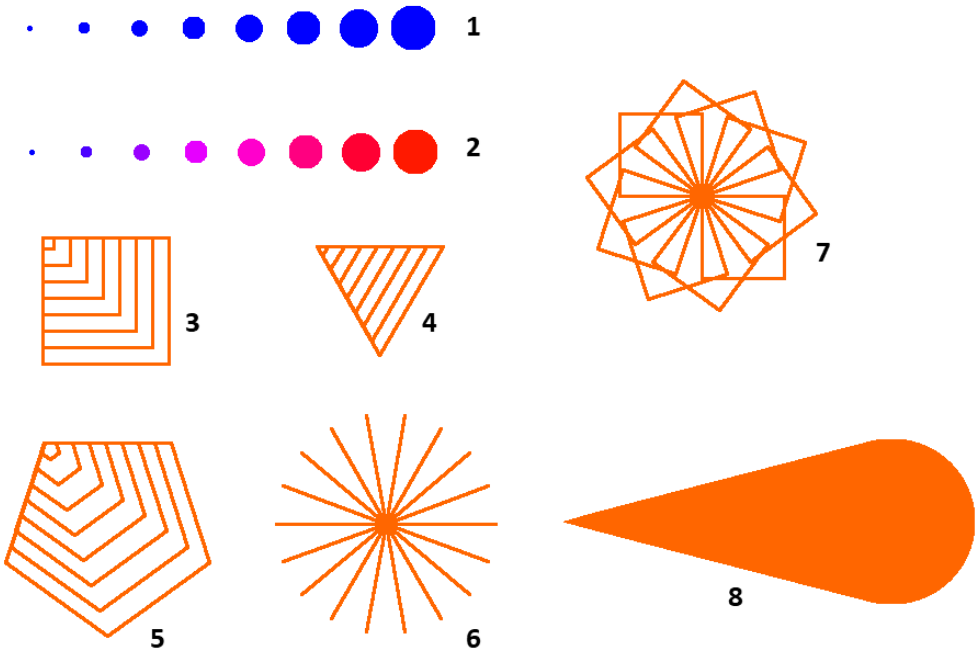
35. What would go in the empty cell in line 2?
    A. x – y
    B. x
    C. y
    D. y – x

36. What would go in the empty cell in line 4?
    A. 1
    B. x
    C. y
    D. -1

Programming problems:

(Refer to Appendix A for links to demo videos of some of these problems.)

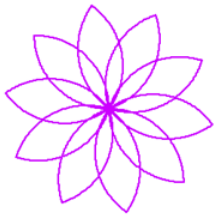
Pen-art-1:
Write scripts to draw the designs shown in the collage below:



Pen-art-2:
Write scripts to draw the designs shown in the collage below:

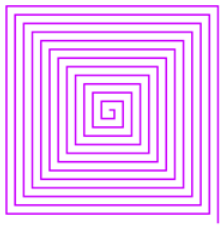
9

10



11

12



13

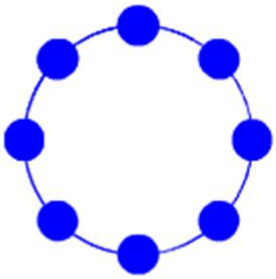
14

Pen-art-3:

Write scripts to draw the designs shown in the collage below. The scripts should keep the sprite moving continuously drawing the same pattern over and over again.

15

16



17

Double-pen:

Write a program that allows you draw with a double color as shown below:

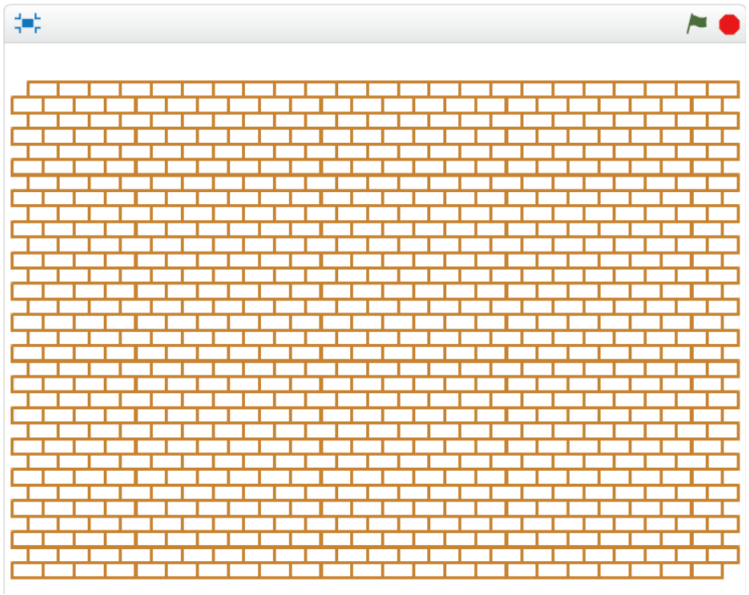
Repeat patterns-2:

Modify "repeat patterns-1" program and create procedures for “square wave”, “staircase”, and “point star”.

Wall of bricks:

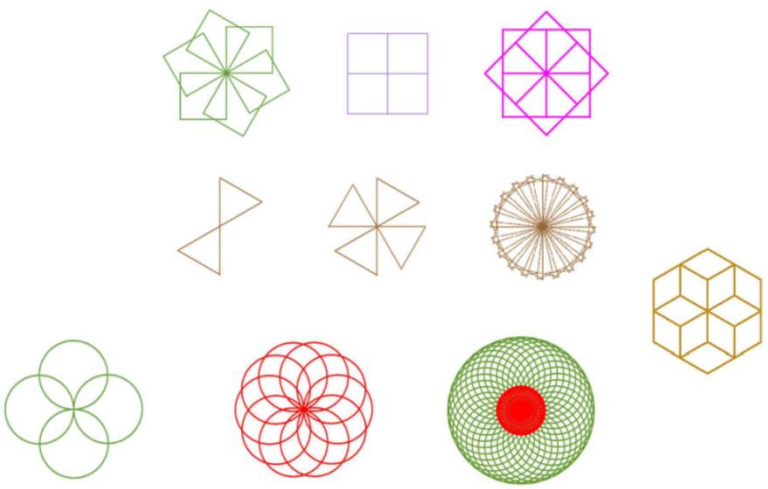
Build a wall of rectangular bricks:

a. Write a script to draw a single 20x10 "brick" in the lower left corner of the stage when the green flag is clicked, something like this:
b. Now modify your code to build a full row of bricks across the entire length of the stage. Use a loop to keep your code as concise as possible. Remember that the stage is 480 pixels wide.
c. Now that you can build a row of bricks, add code to build a second row above the first row. The bricks in the second row should share a long edge with those in the first row, but should be "offset" so that the ends of the second row bricks are at the middle of the first row bricks.
d. Finish off the wall by building alternating rows all the way to the top of the stage. Remember that the stage is 360 pixels tall. Your final wall should look like this:



Flowers using nested looping:

First draw the basic shapes (square, triangle pentagon, circle, etc.) using repeat. Then draw the flower designs (shown below) using nested repeat.



Etch-a-sketch-1:

Write an Etch-a-sketch program:

a. Green flag brings the turtle at the center.
b. SPACE key clears the screen.
c. "D" puts the pen down.
d. "U" pulls the pen up.
e. Arrow keys move the turtle around the screen.

Shapes-1:

Write a script that draws a square when the number 1 is pressed on the keyboard. The sprite should say "square" while it is drawing. It should stop saying "square" once it has finished drawing. The sprite should hide when it is not drawing. Pressing the space bar should clear the drawing. In a similar fashion, write scripts to draw an equilateral triangle, a diamond, a pentagon, a parallelogram, and a 5-point star when keys 2, 3, 4, 5, and 6 are pressed. Each shape should have a different color and a different pen size.

Shapes-2:

Modify the above program such that you have new custom blocks to draw all the shapes. The custom blocks will not take input.

Etch-a-sketch-2:

Modify the Etch-a-sketch-1 program such that combination of keys moves the turtle at different angles. For instance, if you press left and down arrows simultaneously the turtle should move in the southwest direction. Also provide a slider to change brush thickness and a palette of colors to choose from.

Shapes-3:

Modify the "shapes" program such that all custom blocks except parallelogram will accept appropriate input. For example, "diamond" will have 2 inputs: length and angle.

Polygon:

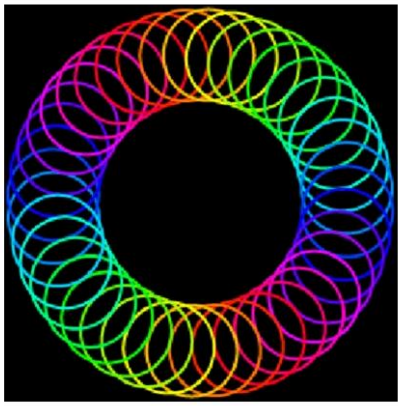
Write a program in which you have a new block called "polygon" which will take 2 inputs: # of sides and length of each side. This new procedure should be able to draw any regular polygon.

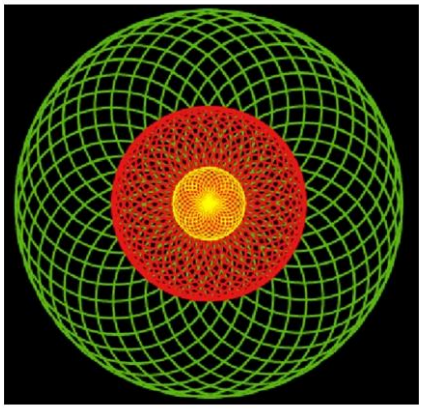
Jump:

Write a custom block “Jump” that takes “height” as input. This block will make the sprite jump up and down. Use “glide” for the jump. (Tip: You may want to use the “x position” and “y position” variables.)

Circle designs-2:

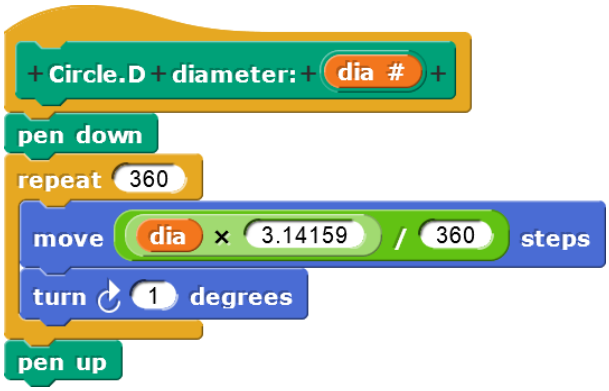
Draw the following circle designs:



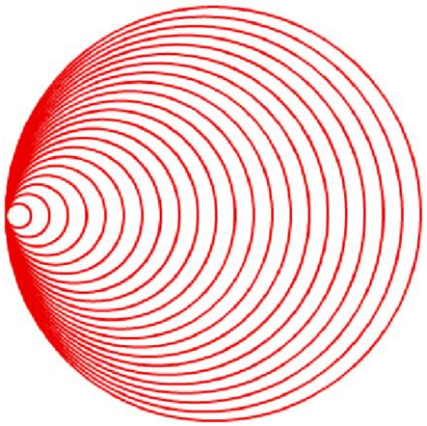


Cone of circles:

See below a procedure for drawing a circle of diameter d.

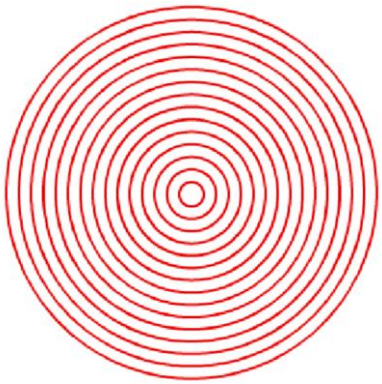


Using this procedure draw this design:



Concentric circles:

Modify the circle procedure such that the circle is drawn with the sprite at the center. Write a script using this procedure to draw concentric circles as shown below:



Rectangle grid:

Using the user-supplied dimensions of a single cell (height and width) draw a rectangular grid MxN such that maximum possible cells are drawn to fit the Snap screen. Then stamp a centered image of the sprite inside each cell. For example, if the cell size is 300x400 only one rectangle should be drawn since only one cell can fit on the screen.

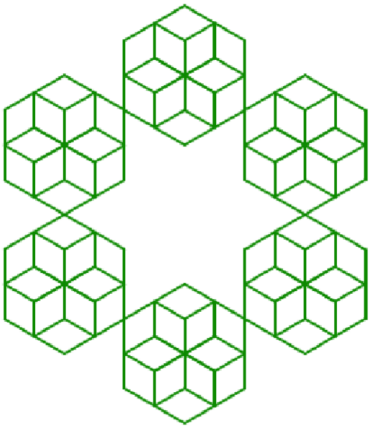
Arithmetic functions:

Write the following arithmetic functions:

a. Check if the input is an integer (and not a fraction). Return True or False.
b. Check if the first input is divisible by the second input. Return True or False.

Hexdesign:

The basic shape in the design below is a hexagon. Using nested looping, see if you can draw the design.



Marathon:

This is a game in which you take a sprite out for a run by using the arrow keys. The sprite leaves a trail behind (by drawing a thick line) which it must not touch while running. As soon as the sprite touches the line the game is over. The program measures the distance run by the sprite and compares it with the previous record.

Notes and guidelines for programming problems:

Repeat patterns-2:
Put each script (that draws a specific shape) into a new custom block.

Wall of bricks:
The problem description itself gives plenty of hints and guidelines. This program is mainly based on the use of nested loops. The innermost loop will draw a single brick. The next outer loop will draw a single row of bricks. And so on.

Flowers using nested looping:
All the designs shown in this problem involve drawing the basic shape a number of times. The trick is to turn a little before drawing each subsequent shape. For example, the first shape in the first row shown above consists of six squares drawn after turning 60 degrees every time.

Etch-a-sketch-1:
This program allows free-form rectilinear drawing using arrow keys.

Shapes-1:
Have a separate script for each shape and invoke each of them using the key-press event. Saying the shape's name while drawing can be achieved using the SAY command without the time input. This command keeps the text bubble on the screen until the next SAY command is run. (Hint: No text bubble appears if a blank input is given to SAY.) Drawing the shapes is a matter of using some school geometry. For example, the TRT (turtle round principle) tells us this: "If the Turtle returns to the exact same spot and faces the same original direction the sum of all turns is 360 (or a multiple of 360)." This implies, for instance, that a triangle can be drawn by turning 3 times by 360/3=120 degrees each.

Shapes-2:

Convert each script into a custom block. So, there would be a "Triangle" block that draws a triangle, and so on.

Etch-a-sketch-2:

In order to allow multiple simultaneous key-press events, we need to use the "key pressed" condition (under the "sensing" category) inside a forever loop and use Boolean operators to combine multiple conditions.

To control the pen's thickness, create a slider variable with min and max values. Then, watch this variable continuously (in a forever loop) and apply its value to the pen thickness.

Finally, create a color palette by simply having a circle sprite for each color. When the user clicks on any of these sprites, use "set pen color" to apply the sprite's color to the pen.

Shapes-3:

Modify the "shapes-2" program to supply appropriate input parameters to each of the new custom blocks. For instance, the "Triangle" block would take "size" input which determines the length of each edge.

Polygon:

According to the TRT principle a regular polygon of N edges will involve N turns each 360/N degrees. For instance, to draw a polygon of 5 sides (pentagon) the turtle will turn 5 times with each turn being 360/5=72 degrees.

Jump:

Use "glide" to make the jump appear smooth. As suggested in the problem, you will need to use the properties "x position" and "y position" to ensure the jump can occur where the sprite currently is.

Circle designs-2:

The first step is obviously to write a "circle" procedure (custom block). The first design (kind of bracelet) involves drawing circles repeatedly while moving along a circular path.

The second design consists of 3 identical patterns each using circles of varying size (and color). Each pattern involves drawing circles repeatedly around a point, i.e. you draw a circle, turn a little, draw a circle, turn a little, and so on. Each turn is such that the total comes to 360 degrees.

Cone of circles:

In this design, you need to draw circles repeatedly while increasing the diameter every time. Use variables for this purpose. See the algorithm below:

```
Count = 1
Diameter = 20
Repeat 20
      Circle (Diameter + (10*Count))
      Count = Count + 1
End-repeat
```

Concentric circles:

This program involves two challenges. First is to have a "circle" procedure that draws around a center point. We already have a circle procedure that draws a circle of diameter D. We can modify it slightly as below:

```
Shift left by D
Draw circle of diameter D
Shift right by D
```

The second challenge is to draw the given pattern of concentric circles. This can be achieved easily by repeatedly drawing circles of increasing diameter. Use variables for this purpose.

Rectangle grid:

Let's say Snap's screen area is H (height) by W (width)
Each rectangle is h (height) by w (width)
# of rectangles in a row = W/w
# of rectangles in a column = H/h
# of rectangles in the grid = W/w * H/h

Algorithm to draw the grid:

```
W = 480
H = 360
# of rows = nr = H/h
# of columns = nc = W/w
# of rectangles = R = nr * nc
I = 0 (1st row)
Repeat nr
      Move to beginning of I'th row (x=-W/2,y=-H/2+(i*h))
      Repeat nc
            Draw rectangle
            // place a face in each rectangle
            Move to center of rectangle (x+=0.5w, y+=0.5h)
            Stamp
            Move right (x+=0.5w, y-=0.5h)
      End-repeat
      I = I + 1
End-repeat
```

The next challenge is to set the sprite size such that it fits the rectangle of dimensions h x w. Here is a possible algorithm for that:

First find out dimensions of the sprite (click on "costumes" tab):

```
Height of sprite = ht
Width of sprite = wd

If h>w
      p = 100*(w/wd)
Else
      p = 100*(h/ht)
End-if
Set size p
```

Arithmetic functions:

How do we check if the input N is a whole number and not a fraction? One way is to convert N to an integer using the "round" operator. For example, round(3.6) gives 4. You can then compare this with N; if they are equal, N is an integer.

Checking divisibility of M by N can be done using the following algorithm:

```
Use the "mod" operator to find the remainder of M/N.
If the remainder is 0, M is divisible by N, else it is not.
```

Hexdesign:

Use triple nesting: the innermost loop will draw a hexagon, the next outer one will draw a 6-petal flower of hexagons, and the outermost will draw the design of 6 flowers.

Marathon:

Use "polling" of key-press events to move the sprite, which allows you to completely stop the movement when the game is over. The sprite will draw a contiguous thick line. To sense touching this line, add a color blob at the nose of the sprite and use the condition "color touching color". Measuring distance is straightforward. Snap retains values of variables even after closing projects, so you can maintain the previous record in a variable.

Unit 8: Additional CS Concepts

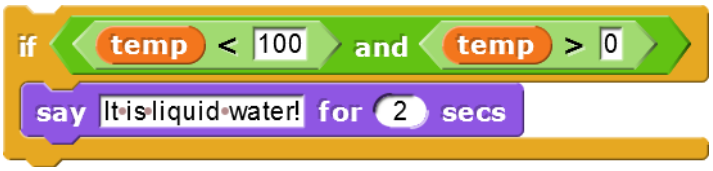
We don't stop playing because we grow old; we grow old because we stop playing.
– George Bernard Shaw

Concepts practiced in this Unit

- All concepts of previous units
- Procedures with return values
- Recursion
- List variables
- Logic operators (AND, OR, NOT)
- Conditional statements (nested IF)
- Mapping random numbers to a set of things
- Mouse events (polling)
- User input validation
- Variables - local/global scope
- Using variables as gates
- Variables - as timer

Concept Review Questions:

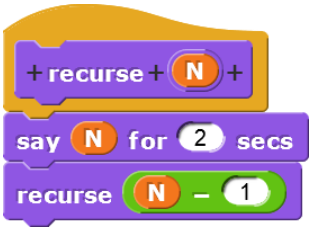
1. For what value of "temp" will the sprite say "It is liquid water"?



   A. -10
   B. 20
   C. 100
   D. 0

2. See the recursive procedure below. If you call it as

what will happen?



   A. The sprite will count down from 50 forever.
   B. The sprite will count down from 50 to 0 and then stop.
   C. The sprite will say 50 forever.
   D. The sprite will say N, N-1, N-2, and so on, forever.

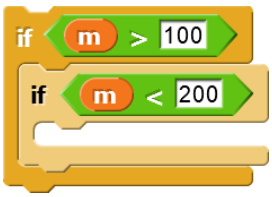
3. Read the following script. What does it do?

   A. The sprite will keep moving and will stop when it touches orange color.
   B. The sprite will keep moving only as long as it is touching orange color.
   C. The sprite will stop moving when it becomes orange.
   D. The sprite will not move at all.

4. See the script below. For what value of m will the sprite NOT say "Hurrah"?

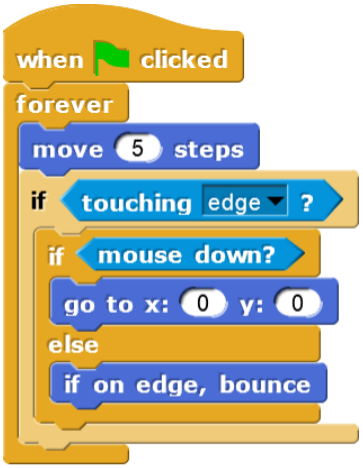
   A. 0
   B. 150
   C. 30
   D. 175

5. If we wanted to convert the following nested IF statements into a single IF statement, which operator can we use?



A. The AND operator
B. The OR operator
C. The NOT operator
D. Both the AND and OR operators

6. What will happen when you run the following script?



A. Every time the sprite touches an edge it will bounce and continue moving.
B. Every time the sprite touches an edge it will jump to the center only if mouse is NOT down.
C. The first time the sprite touches an edge it will jump to the center and the second time it will bounce.
D. Every time the sprite touches an edge it will jump to the center only if mouse is down.

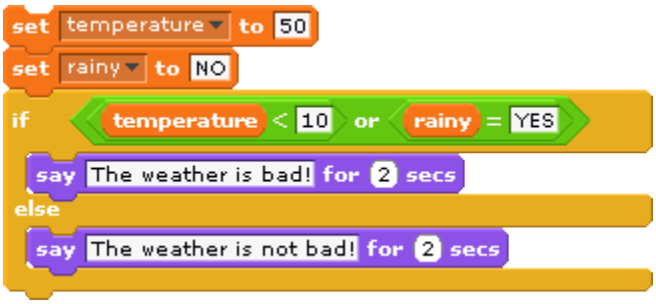
7. What will happen if we run the following script?

A. The sprite will say "You win!!"
B. The sprite will say "You lose!!"

8. What will happen if we run the following script?

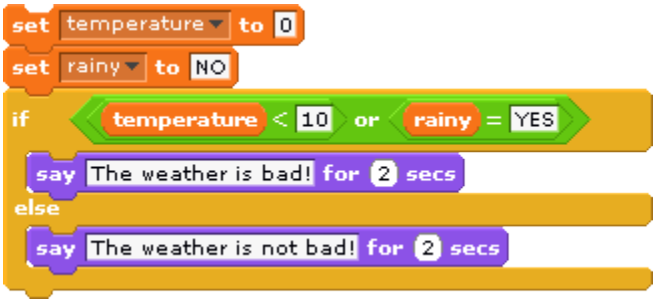
A. The sprite will say "You win!!"
B. The sprite will say "You lose!!"

9. What will happen if we run the following script?



A. The sprite will say “The weather is bad!”
B. The sprite will say “The weather is not bad!”

10. What will happen if we run the following script?



A. The sprite will say “The weather is bad!”
B. The sprite will say “The weather is not bad!”

11. Will the following scripts do the same thing?

A. Yes
B. No

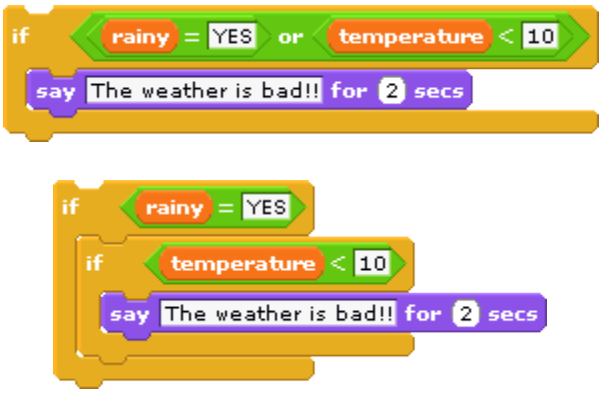
12. Will the following scripts do the same thing?

A. Yes
B. No

13. Will the following scripts do the same thing?

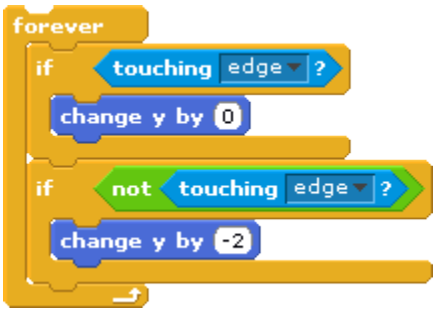
A. Yes
B. No

14. Will the following scripts do the same thing?



A. Yes
B. No

15. In the script below the sprite is expected to fall down continuously until it hits the edge of the screen.



A. This script won't work because the sprite will not stop at the edge.
B. This script is correct and both IF statements are essential.
C. The first IF statement is not required and can be removed.
D. The second IF statement is not required and can be removed.

Programming problems:

(Refer to Appendix A for links to demo videos of these problems.)

Yoda-speak:

(Main concept: Lists) Move the first two words in a sentence to the end of the sentence. This is also known as "Yoda-speak" because the Star Wars character Yoda spoke like that. For instance, instead of saying "This is my home" he would say "my home this is".

Placement of cards-2:

Modify the program such that the layout is random, i.e. every time you run the program the images can occur at different places in the layout. Press SPACE to collect the cards in one spot (maybe a corner) and click Green flag to place them one by one.

Color change-2:

(Main concept: Boolean operators) Take the "color change-1" program as starter code: you wrote this program in Chapter 2 (problem 2). Now, make the following changes:

- To turn black:
  - If touching black AND key 'a' is pressed
- To turn orange:
  - If touching orange OR mouse button is down
- To turn red:
  - If touching red AND key 'b' is not pressed
- To turn blue:
  - Add a variable called “score”
  - It should take a random value in the range -100 to 100 every 2 seconds
  - Turn blue if touching blue AND “score” is positive

Jumping penguin:

(Main concept: Boolean operators) Have 3 sprites: Penguin, ground, and a small platform. Write a "Green flag clicked" script for the Penguin sprite in which:

a. Penguin goes to a random spot at the top of the screen.
b. It falls down under gravity with a small acceleration.
c. It stops falling if it hits the ground or the platform.

Next, write a "When spacebar pressed" script which makes the Penguin make a short jump only if it is stationary (i.e. it is resting on ground or the platform).

Which edge:

(Main concept: Boolean operators) Detect which edge the free-moving ball is touching of the screen. As the ball hits an edge, a variable shows which edge it is (top, bottom, left, right). You could get more fancy and light a bulb next to the edge.

Grading:

(Main concept: Boolean operators) Using the following table, write a program that converts marks into grades. The user will enter his/her marks and the program will display the grade.

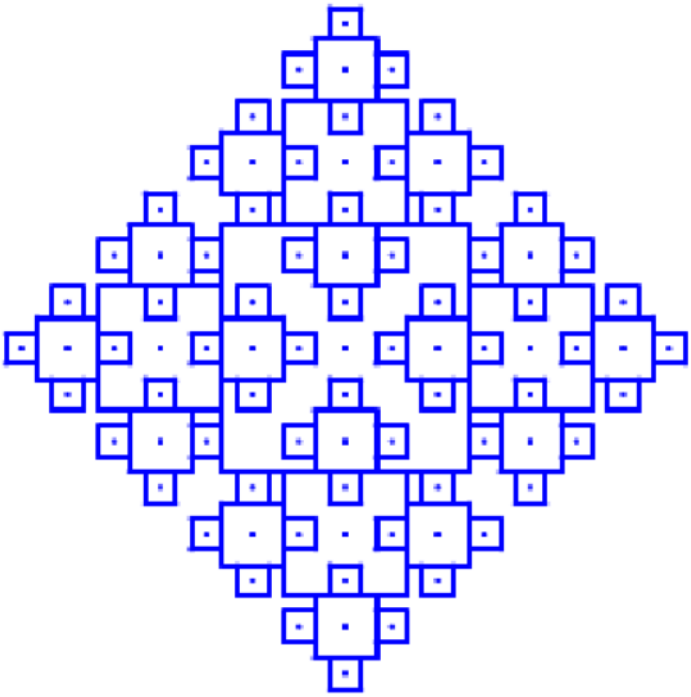
Points	Grade
90 or higher	A
75 or higher	B
60 or higher	C
All others	F

Friends:

(Main concept: Lists) Build a database of "friends" with click-buttons to add (a new friend), check (if a person is friend or not), remove, replace (with a new one), count # of friends, and show/hide (the entire list).

Recursive design:

(Main concept: Recursion) Design the following recursive drawing.



Sentences:

This program creates simple random sentences using the subjects, verbs and objects supplied by the user.

Butterfly-2:

Modify the previous butterfly-1 program as follows: (1) The butterfly leaves the screen occasionally. (2) Add another butterfly. The two butterflies should do their thing independently of one another.

Snap help:

The idea is to present pictures of Snap commands and when the user clicks on a command an information bubble pops up. Also include "sample code" for one or more of the commands, which pops up if user presses SPACE key within a few seconds of clicking on that command.

Notes and guidelines for programming problems:

Yoda-speak:

Step 1: Convert the given sentence into a list of words:

We can use the "split" operator in Snap (under "operators") for this purpose.

Algorithm:
Extract words from the given sentence.

```
Given:
Str: sentence containing words separated by single space

Output:
List of words

Procedure:
Store the extracted words in list L.

Variable L = split( str, whitespace )
```

Step 2: Create a sentence with the first two words moved to the end.

Algorithm:

```
Create a new "Yoda" sentence by connecting words in list L
such that the first two items come at the end.

Given:
L: list of words

Output:
Sentence with first two words moved to the end
```

```
Procedure:

yoda = ""
words = size of L
If (words > 2)
      i = 3
      Repeat words-2 times
            append to yoda item i in L
            append space
            i = i+1
      End-repeat
End-if
If (words > 0)
      append to yoda item 1 in L
      append space
End-if
If (words > 1)
      append to yoda item 2 in L
End-if
```

Placement of cards-2:

In real-life games, the cards need to be placed randomly. We can achieve that if someone can assign positions 1 to 8 one by one to a randomly selected sprite/clone. How do we do that?

Here is one approach which modifies the algorithm used in the earlier Version 1.

```
Numbers = Create a list of numbers 1 to 8
Count = 8
Set Y (Y position of the 1st row)
Repeat 2
      Set X (X position of the 1st card)
      Repeat 4
            I = pick random 1 to Count
            Tell picture card numbers[I] to place itself
            Remove Numbers[I] from the list
            X = X + 100 (X position of the next card)
            Count = Count - 1
```

```
      End-repeat
Y = Y - 140 (Y position of the next row)
End-repeat
```

Color change-2:

These changes are straightforward: you just need to use the Boolean operators AND, OR, and NOT.

For the color "blue", use a variable, say, "N". You will then need to run a separate script that sets the value of N to a random value in the range (-100, 100) every 2 seconds.

Jumping penguin:

To make the Penguin stop falling when it touches ground or a platform, use the following algorithm:

```
Forever:
      IF touching ground OR touching platform
            Change y 0
      Else
            Change y -5
      End-if
End-forever
```

To get some acceleration, use a variable for speed, i.e. for the input to "Change y". Then, you can continuously vary that variable and achieve acceleration.

Which edge:

One way to solve this problem is to break it into two steps:

Step 1:
Using the fact that the ball bounces differently off vertical and horizontal edges, first determine whether the ball has bounced off vertical or horizontal edges. When the ball bounces off vertical edges its new direction is (360 – old direction),

whereas when it bounces off horizontal edges, its new direction follows a more complicated formula. See the following algorithm:

```
If touching edge
      Bounce
      If bouncing off vertical edges
            Ball has hit either left or right edge
      Else
            Ball has hit either top or bottom edge
      End-if
End-if
```

Step 2:
Distinguish within top/bottom and left/right edges.

Once we know that the ball has hit a vertical edge, we need to figure out whether it's the left or the right edge. This can be achieved simply by looking at the value of "X position". If it's negative, it's the left edge; otherwise it's the right edge.

Similarly, once we know that the ball has hit a horizontal edge, we need to figure out whether it's the top or the bottom edge. This can be achieved simply by looking at the value of "Y position". If it's negative, it's the bottom edge; otherwise it's the top edge.

Grading:

We can solve this problem by using Boolean operators and "nested conditionals", i.e. using an IF inside another IF. See the algorithm below:

```
If score = 90 OR score > 90
      Grade is A
Else
      If score = 75 OR score > 75
            Grade is B
      Else
            If score = 60 OR score > 60
                  Grade is C
```

```
            Else
                  Grade is D
            End-if
      End-if
End-if
```

Friends:

This program can be written using the "list" feature of Snap. There are list commands available to directly perform some of the database operations.

For example, the "add" command can be used to add new friends to the database. The "contains" condition can be used to check if a name already exists in the database. The "length of" operator can be used to find the number of friends.

The main challenge is figuring out how to "remove" and "replace" friends. There is no direct way to know where in the list a given name exists. For example, let's say you want to remove "Peter" from the database. You first need to find out where in the list "Peter" is located. Once you know the location number (e.g. 1, 2, etc.), you can use the "delete" command to remove "Peter" from the list.

The following algorithm allows you to "search" for a given name. This algorithm would be useful for both "remove" and "replace" operations.

```
Name = name to be searched
N = number of friends (length of the list)
I = 1
Repeat N
      If I'th item equals Name
            Name found ..
      End-if
      I = I + 1
End-repeat
```

Recursive design:

In recursive designs, the main challenge is to figure out the basic pattern which is repeated over and over again until a terminating condition is reached. In this case, we could use "depth of recursion" as the terminating condition.

The basic pattern in the given design (i.e. with no recursion) is a square with a dot at the center. If you recurse another level, each existing square gets a "child" square (of half the parent's size) on top of each of its edges.

Thus, you could simply use an IF command to check the depth of recursion which would be passed as a parameter. If the depth is 0, only the basic pattern is drawn. If the depth is 1, the program will recurse once. And so on.

Recursing would involve getting in position to draw a smaller square on each edge and then making a self-call with a reduced depth and reduced square size.

Sentences:

First ask the user to enter 5 subjects and save them in a list. Next, ask the user to enter 5 verbs and save them in another list. Finally, ask the user to enter 5 objects and save them in a third list.

Now, to create a sentence, do the following:

- Pick a subject, a verb, and an object randomly from these lists
- Join them to make a sentence

Butterfly-2:

In version 1, we use the "if on edge bounce" command to make the butterfly fly around the screen but always stay in view of the user. In real life the butterfly doesn't always stay in front of the camera! Occasionally it goes away and then reappears some time later. To implement this behavior, we could simply make the butterfly "hide" once in a while as soon as it touches an edge. And then after

a brief but random interval of time, it would become visible again, bounce off the edge and continue flying.

Hiding "once in a while" can be implemented by simulating a die.

Adding another butterfly may appear to be a matter of copying all scripts. This is true; but, there is a catch. You cannot share variables between the two butterflies because then they would behave identically all the time! For example, when the first butterfly slows down (due to the "speed" variable becoming smaller) the second butterfly would follow suit. To make the two butterflies completely independent of each other, make all variables private (use the option "for this sprite only"). Once you make the variables private for the main butterfly, you could simply duplicate it to create the second butterfly.

Snap help:

Each command is a separate sprite.
Use "command-info" list variable that contains information about every command. Each command is a separate sprite with its own number which relates to the location in the list.

When a command image is clicked, it fetches the information from the global "command info" list and presents it.

The second list called "commands" has a list of the associated Snap commands. This list is used to furnish code sample. This is demonstrated by "Forever" in the demo video. When Forever is activated it allows the user to see the sample code if he presses the SPACEBAR within 5 seconds.

Appendix A: Program Demos

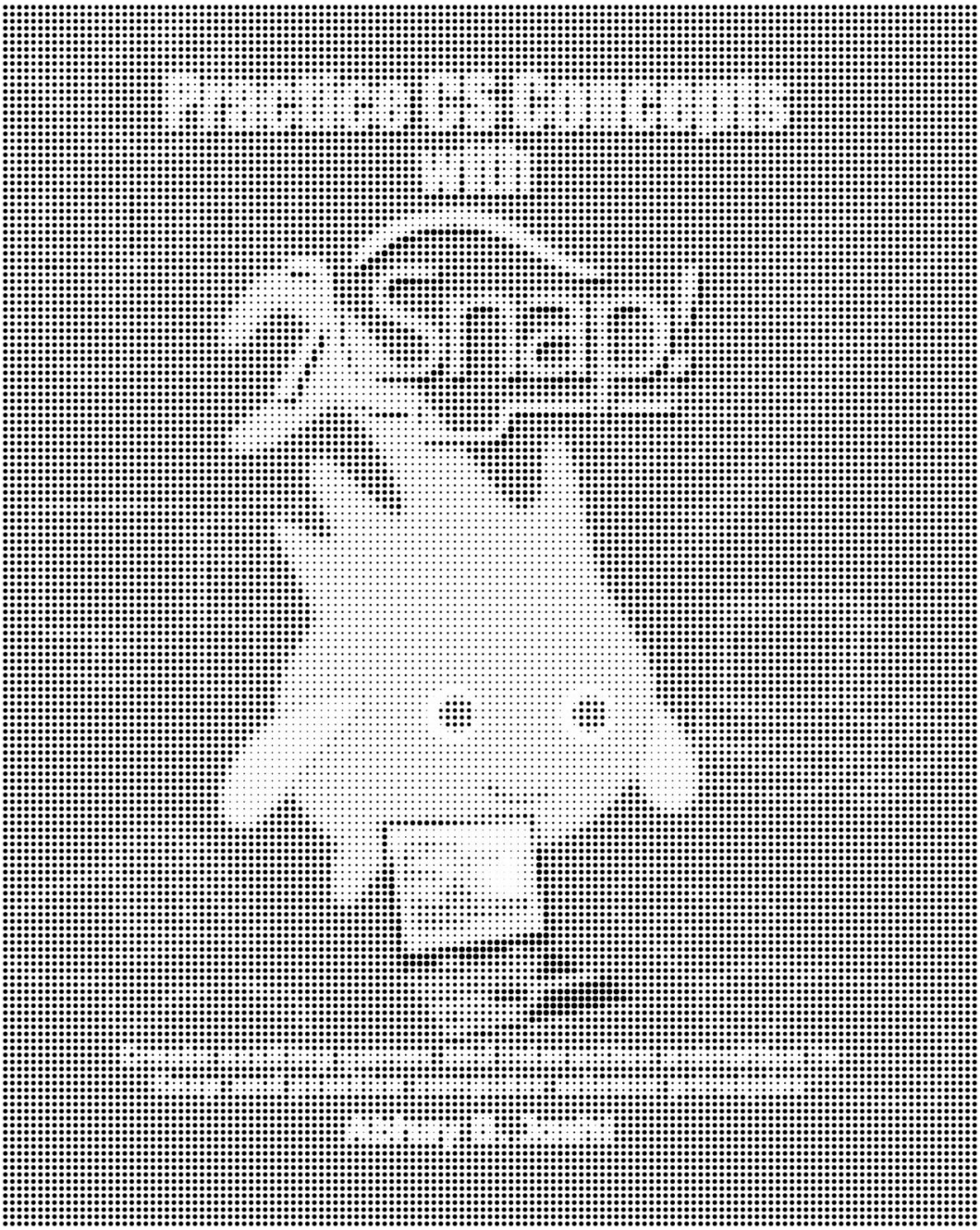
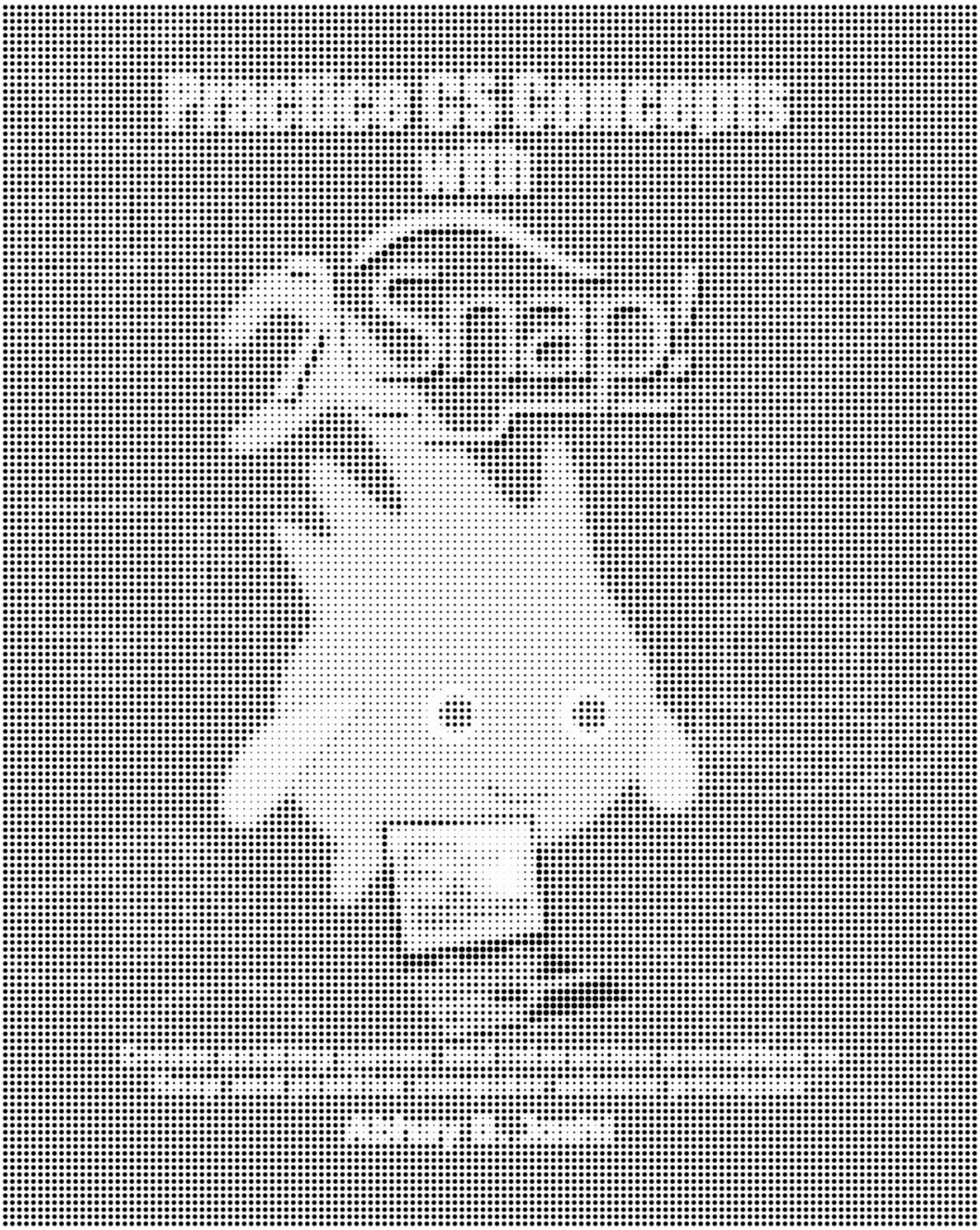
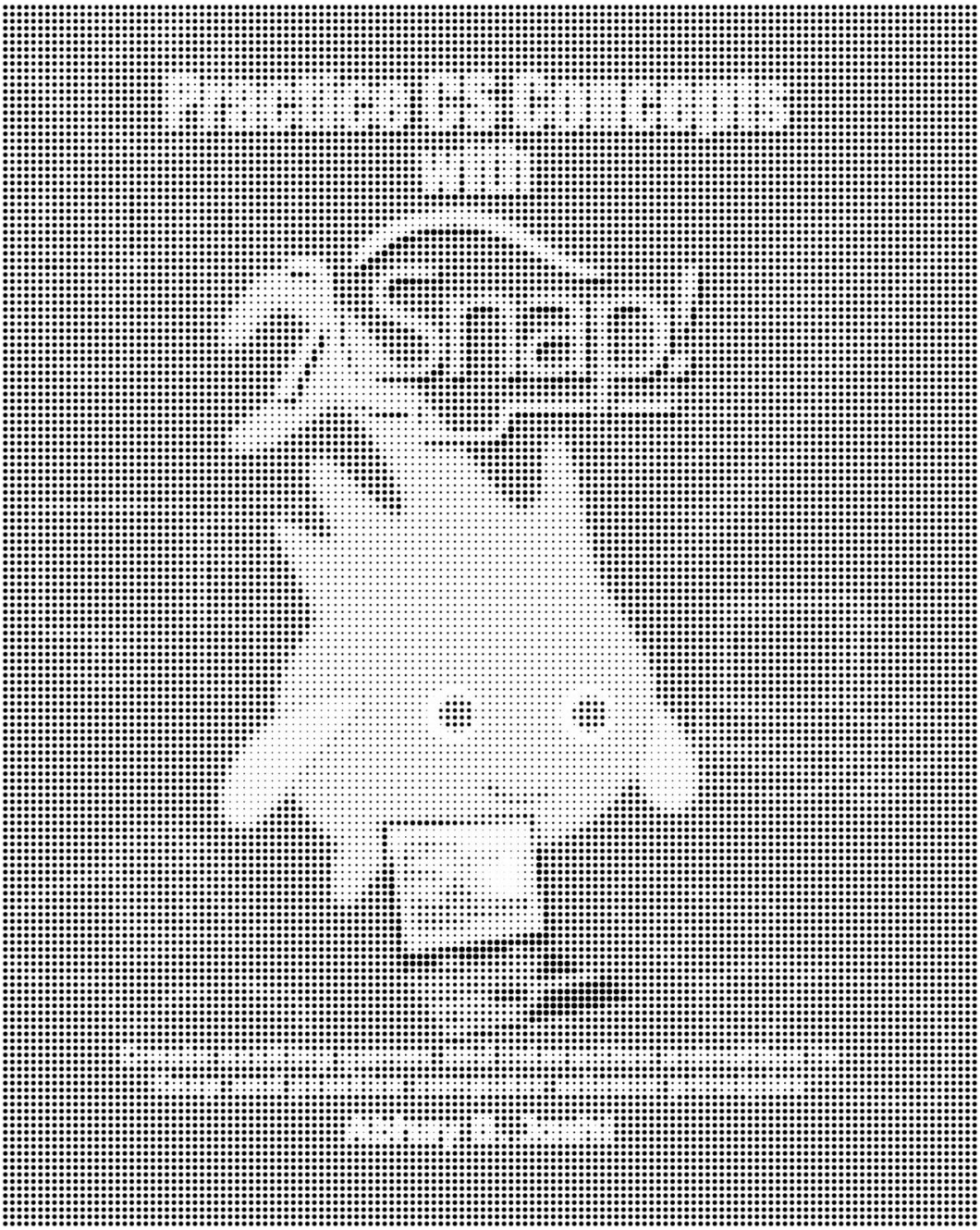
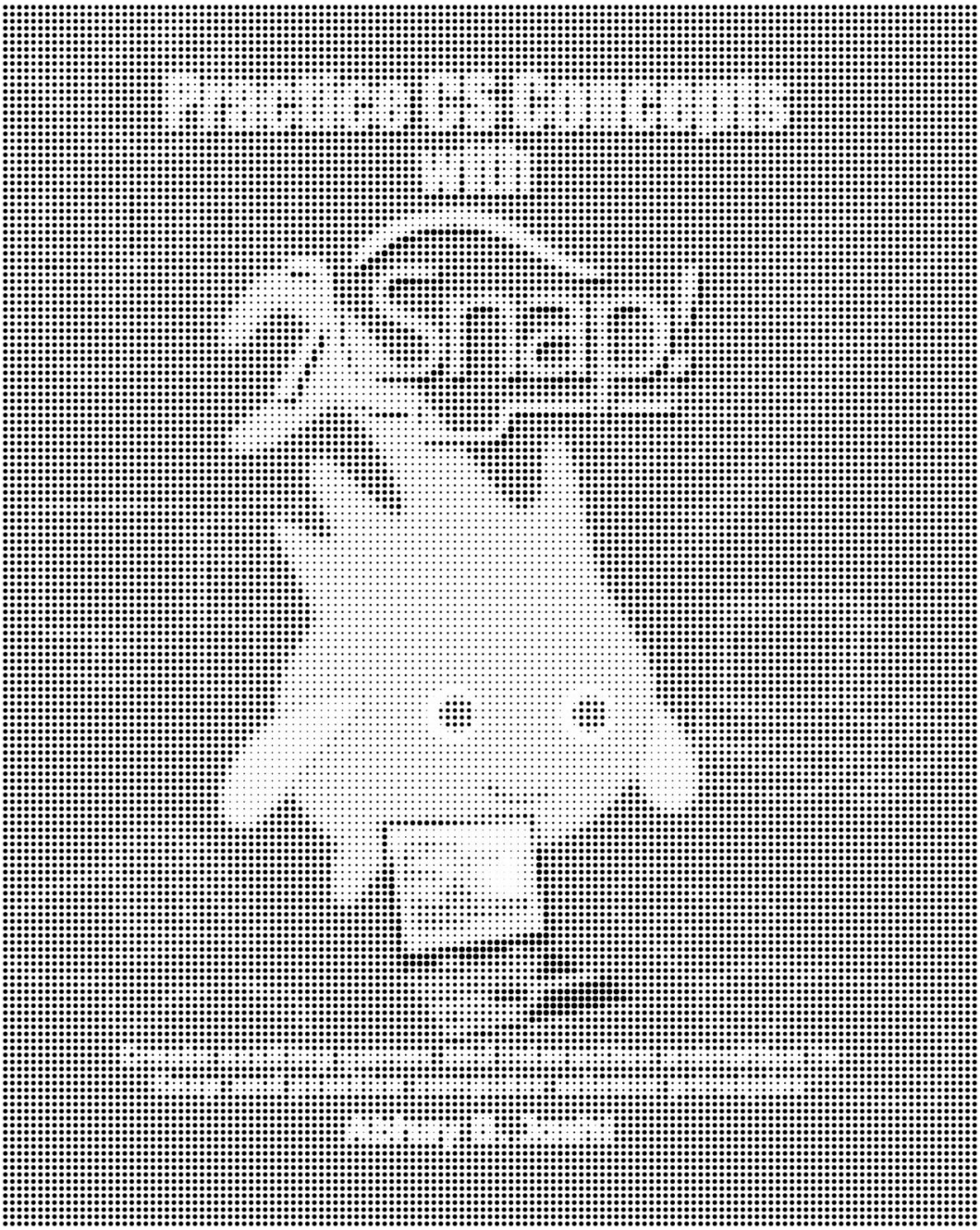
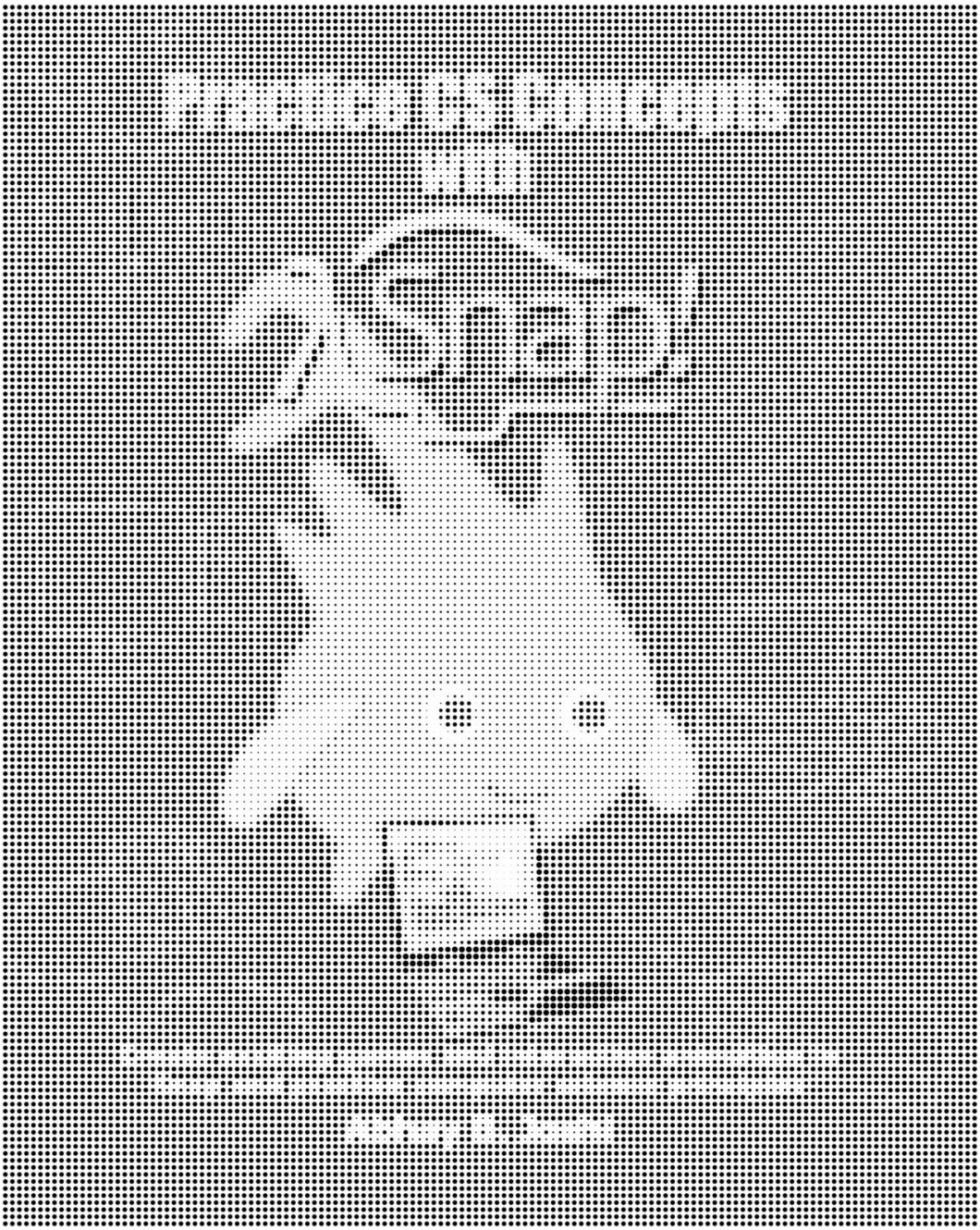
Now more people are doing work that requires individual decision-making and problem-solving, and we need an educational system that will help develop those skills. – Seymour Papert

Demo videos of the programs:

You could view these video demos to better understand how each program is expected to behave. Note that, the demos depict only one possible behavior; you should feel free to use your creativity to modify the look and feel of your program. But, make sure your program meets the basic requirements of the respective assignment.

http://abhayjoshi.net/spark/snap/book0s/videos.htm

Concepts used in the programs:

Download the following Excel file which gives in a table format which CS and Snap concepts are expected to be useful for each programming assignment. Note that this is not a strict requirement: You might find different approaches to solve the problems, which require additional or different concepts.

http://abhayjoshi.net/spark/snap/book0s/concepts.xlsx