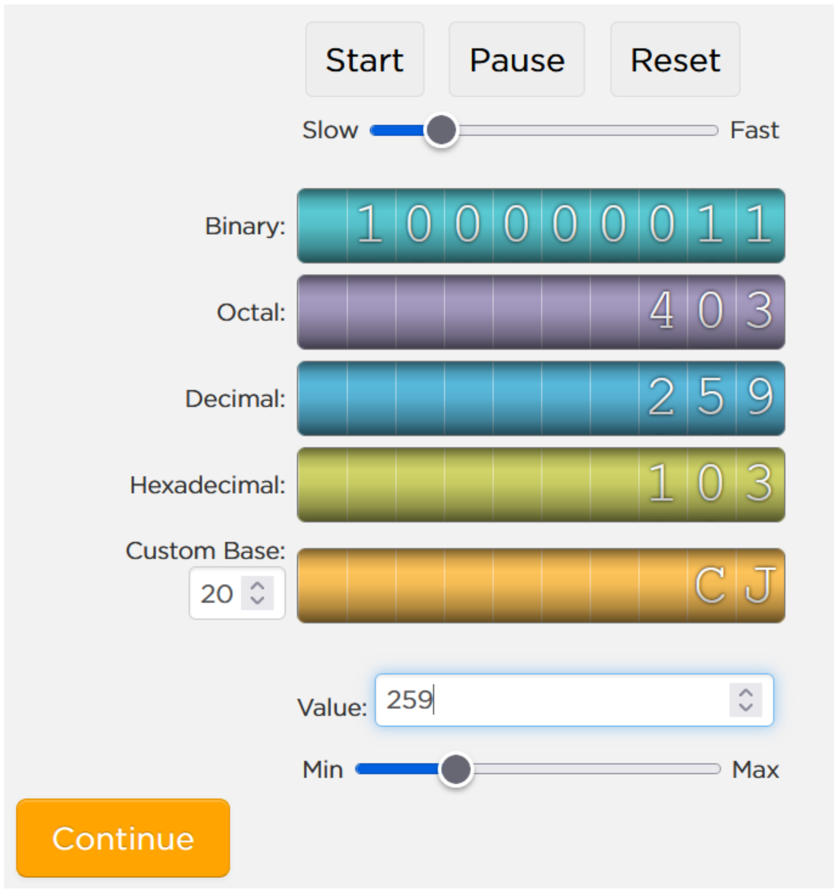
Base Converter and Counter

This program allows you to convert numbers from one base to another. We will allow bases 2 (binary), 8 (octal), 10 (decimal), and 16 (hexadecimal). It is a kind of educational tool to further our understanding of these number systems.

We know that digital computers use the binary system in their hardware operations. Number systems are a fundamental part of Computer Science Education. This program was inspired by the Computer Science Principles course designed by College Board and we have borrowed some of the features of the Code.org demo program as shown below:



Binary: 1 0 0 0 0 0 0 1 1
Octal: 4 0 3
Decimal: 2 5 9
Hexadecimal: 1 0 3
Custom Base: 20 — C J
Value: 259

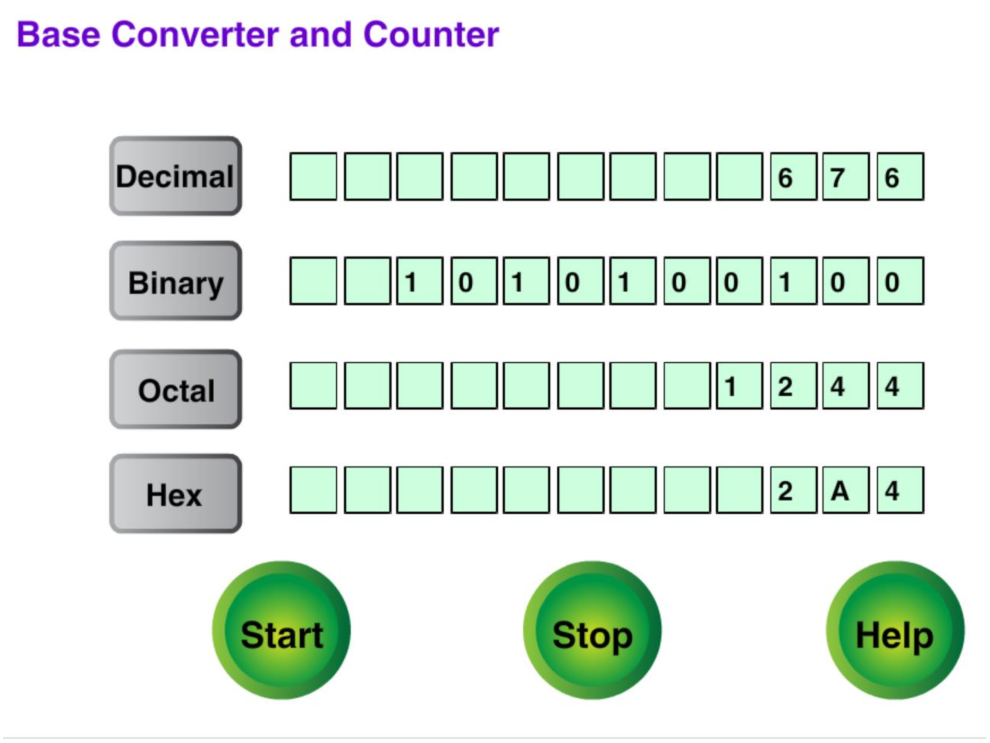
If you want to play with my final program to get a feel for this game, click the link given at the end of the article. Try not to peek at the scripts yet, since we want to design them ourselves below. Click here to download all program files.

How to run the program:

1. Click Green flag. Click 'help' to view instructions.
2. Click any of the base names to enter a number of that base. The program then automatically shows that number in all 4 bases.

3. Click 'start' to start counting. The program starts counting from the current number.
4. Click 'stop' to stop counting.

Here is how my final program looks like:



Scratch and CS Concepts Used

When we design this program, we will make use of the following Scratch and CS concepts. I assume that you are already familiar with these concepts.

- Algorithms
    - Abstraction
    - Using algorithms
    - Designing new algorithms
    - Pseudo-code
- Animation using costumes
- Arithmetic
    - Expressions
    - Basic operators (+, -, *, /)
    - Advanced operators: mod, floor
- Concurrency
    - Synchronization
    - Synchronization using broadcasting

    - Synchronization using variables
- Conditional statements:
    - Conditions: YES/NO questions
    - Relational operators (=, <, >)
    - Conditionals (IF)
    - Conditionals (If-Else)
    - Conditionals (nested IF)
    - Boolean operators (and, or, not)
- Data structures – list
    - List operations
    - List traversal
- Data types – strings
    - String operations (join, letter, length of)
    - String traversal
- Divide and conquer (program design technique)
- Events
- Looping (iteration)
    - Looping - simple (repeat, forever)
    - Looping - conditional (repeat until)
- Motion (Scratch and Snap)
    - Motion - absolute
- OOP
    - Clones
    - Clones differentiation: using private id
- Procedures
    - Built-in
    - User defined (custom)
    - Simple
    - With inputs
- Program output
    - Text
- Sequence
- STAMP - creating images
- Stopping scripts
- User input
    - Text
    - Click buttons
    - Input validation
- Variables
    - Simple
    - Properties (built-in)
    - Local/global scope
- XY Geometry (Scratch and Snap)

Do you want to check out a working Scratch version of this program? Run the file mentioned at the bottom of this article. I encourage you to explore the program and its various features. But don't look at the Scratch scripts yet; you want to design this program yourselves!

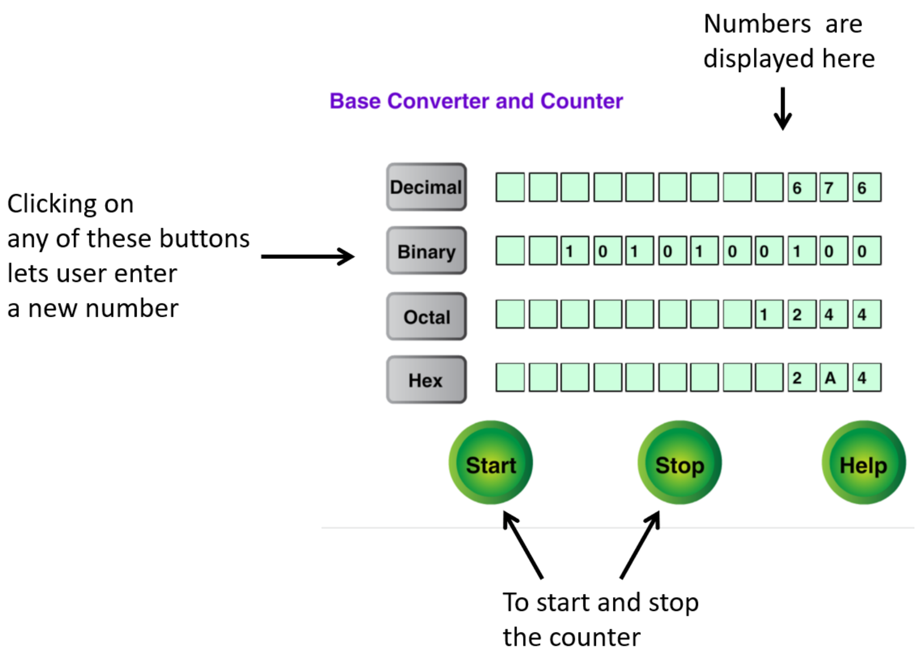
Program Design

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

High Level Design:

This is where we take a step back from the computer (literally!), analyze the problem in our mind (and on a piece of paper if necessary), and break it down into multiple smaller ideas which can be programmed separately.

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


Numbers are displayed here
Base Converter and Counter
Clicking on any of these buttons lets user enter a new number
Decimal
6 7 6
Binary
1 0 1 0 1 0 0 1 0 0
Octal
1 2 4 4
Hex
2 A 4
Start
Stop
Help
To start and stop the counter

It probably makes sense to first get a single counter working – which involves displaying a number in a row of green squares. We can then replicate that technique to have four counters. Next, we could design algorithms to convert between one base to another. Finally, we will add the buttons to implement counting.

Let's get rolling with these various ideas one by one.

For each idea, I will state the requirement and propose a “design” that will allow us to program this idea. You should try to think of your own design and solution before reading the design that I propose.

Feature Idea # 1

Create a single 12-digit counter.

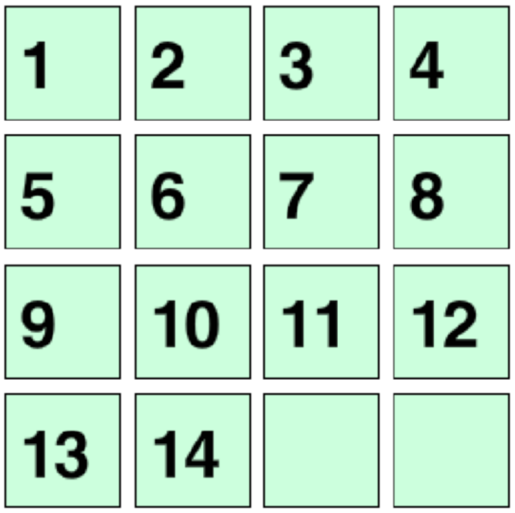
Design:

As always, it is good to consider “reuse” of some existing code that we designed ourselves in the past. Refer to the program "Number table" in my collection of Scratch projects:

http://www.abhayjoshi.net/spark/scratch/blog/number-table.pdf

It explains how to design an NxN matrix with numbers. I think we can take this as “starter code” and get our feature idea #1 implemented.

Let us first study this program carefully. In its existing avatar, the program asks for a number N and then displays numbers 1 thru N in a matrix format. Here is an example for N = 14:



The program does this by using two sprites: a ‘square’ sprite stamps itself to create multiple instances, and a ‘number’ sprite uses the stamp command and costumes to create different numbers on top of the squares.

When you study this program and read its code, you will realize that we will need to make several changes to this program to reach our objective. Let’s consider these changes one by one:

First, we will modify the program to only draw 1xN table, i.e. a single row of N squares.

Step 1: The first modification would be to only display a row of squares instead of a matrix.

This can be accomplished by simplifying the "Draw Table" script of the 'square' sprite.

Next, let us think about the number part. In our project (base converter) the counter frame (i.e. just the sequence of squares) would be static (i.e. drawn only once with no future change), but we need the ability to frequently change the numbers – sometimes just individual digits. And so, the idea of "stamping" is not suitable for the numbers.

Step 2: The next modification in the program would be to use "clones" instead of "stamp" for the numbers.

This will require the following changes in the code for the 'shownum' sprite:
- Use "create clone" instead of "stamp".
- Ensure that the "show number" script runs only for the parent. For this we will require a private "id" variable to distinguish parent from all the clones.

Next, let us consider the relationship between the squares and the digits displayed in them. in the original program, 'square' and 'number' sprites are rather tightly associated. The 'square' sprite not only displays the squares, but also calculates the position where the numbers should appear (see the "Place Number" script). We will simplify the 'square' sprite such that it will only create the array of squares and nothing more. We will move the logic of creating numbers to the 'logic' sprite.

Step 3: Make 'square' and 'number' sprites independent of each other.

This will require the following changes in the code:
- Move the "place number" and "show array" scripts to the sprite 'logic'.
- Use a new variable "clonesCreated" to ensure clones are created only once by the parent. That way, future updates to the numbers will not require new clones, but just costume change.
- We will keep the counter blank upon creation, i.e. no number will be displayed.

If you want to see how the modified program would look after all these changes, check out my program below:

File: single-counter.sb2

Feature Idea # 2

Make it a decimal number counter.

Design:

Now that we have one counter, we will call it the “Decimal” counter, and have it display whatever decimal number the user enters.

Step 1: Provide a “Decimal” button which the user can click to enter a new number to display in the counter.

This change is straightforward. The button will simply send a broadcast message when it is clicked. The ‘logic’ sprite will process it. (See below)

Step 2: The counter should display the current decimal number.

At this juncture, we will make our counter of fixed length with length = 12. We can use a 12-letter-long string variable to save user input and display each digit in the counter.

As hinted earlier, we should *create* the number clones only once, and then use costumes to display different digits. For this, we will need to associate each clone with its location in the counter. The easiest way is to use the x, y coordinates of each square. Each clone will have private variables called ‘row’ and ‘column’ that will save its respective x, y values during clone creation. To simplify, we will convert the x, y values to integer using the “floor” operator.

Future updates to the numbers will be initiated through the same broadcast “show number” but it will now be processed by the respective clones by comparing the x, y values. We will use the special letter ‘Z’ to designate a blank square. So, if the digit is ‘Z’, the associated clone will hide.

On the ‘logic’ side, we will need a new script ‘decimal clicked’ that will build the decimal string and then cause it to be displayed through the same “Place number” custom procedure.

If you want to see how the modified program would look after all these changes, check out my program below:

File: decimal-counter.sb2

Feature Idea # 3

Add another counter with a different base, say hexadecimal.

Design:

Let’s see if we can duplicate the counter functionality to display two counters: one showing the decimal number and the other showing its hexadecimal equivalent.

Step 1: Duplicate the decimal counter and its click-button.

As we have designed above, each clone is uniquely identified by x, y values of its square box, which in turn are dependent on the variables ‘startx’ and ‘starty’. So, to create another counter, we just have to use a different ‘starty’ value and then repeat calls to ‘draw table’ and ‘show array’ scripts. Adding a click button next to the second counter and replicating its ‘when clicked’ script would complete the picture.

Step 2: Allow user to enter hexadecimal numbers for the second counter.

Since hexadecimal numbers use digits between 0 and F, we need to add costumes for A, B, C, D, E, and F to the ‘shownum’ sprite.

Next, we will include conversion routines between decimal and hexadecimal. The idea is as follows: when user enters a decimal (or hexadecimal) number, our program should display it in both decimal and hexadecimal formats. For this, we will need algorithms for conversion between these two bases. While we are at it, we will design routines that can convert decimal to any other base, and vice versa.

Step 3: Write a custom procedure that converts a decimal value to any base from 2 up to 16.

Dipping into our Computer Science knowledge, we know that the basic idea in converting a value (decimal) to any base is to continually divide by that base until we cannot divide anymore. The remainder of each division is accrued to the output.

Here is an example: let’s convert 11 to base 2.

```
11/2 results in 5 with remainder 1
5/2 results in 2 with remainder 1
2/2 results in 1 with remainder 0
Hence, the binary output is: 1011
```

Since hexadecimal numbers contain letters, we will use an array to fetch actual digits.

```
digits = [ '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F' ]
```

Algorithm Decimal to Any:

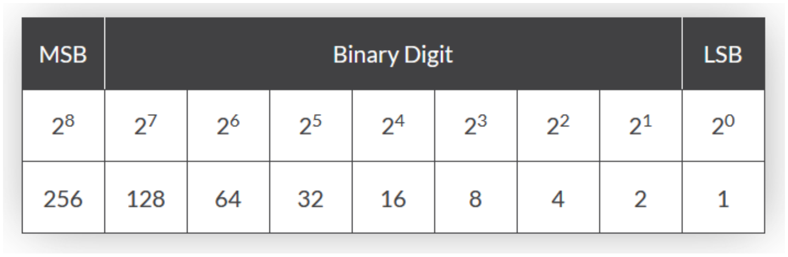
```
Input: decimal value, base
Output: string representing the converted number
Steps:
done = False
any = empty string
```

```
repeat until done:
    rem = remainder of decimal / base
    any = concatenate digits[rem] with any
    decimal = integer division of decimal and base
    If (decimal == 0):
        done = True
    Else if (decimal < base):
        any = concatenate digits[decimal] with any
        done = True
    End if
End repeat
Return any
```

Step 4: Write a function that converts a number in any base from 2 up to 16 to a decimal value.

Once again, dipping into our Computer Science knowledge, we know that the basic idea is to read the number right to left from its least significant digit, compute the digit's place value (digit x its place weight), and accrue it into the total value.

Let us take an example and convert a binary number to decimal. The following table shows the 'place weight' of each position. Starting with LSB where the weight is 1, it grows in powers of 2 as you move to the left.



MSB	Binary Digit							LSB
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

Thus, value of binary 10101 would be 1x2^4 + 0 + 1x2^2 + 0 + 1 = 21.

As before, we will fetch digits from an array:

```
digits = [ '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F' ]
```

Algorithm Any to Decimal:

```
Input: number, base
Output: decimal value

Steps:
placeWeight = 1
for every digit d from right to left:
      d's value dValue = its position in 'digits' array
      value = value + (dValue x placeWeight)
      placeWeight = placeWeight x base
End for
Return value
```

For the highlighted line above, there is no straightforward way in Scratch to obtain the position (index) of an item in a list. So, we will write a separate custom block for that as below:

```
Algorithm index:
Input: c (single letter), digits (list of letters o thru F)
Steps:
I = 1
Traverse 'digits' array
     Compare c with item at I in 'digits'
     If equal, stop repeat
End repeat
Return I-1
```

Save as Program Version 1

Before continuing to the next idea, make a copy of your project with a new name. This way, you have a backup of your project that you can go back to if required for any reason.

Compare your program with my program in the file below.

File: baseconverter-decimal_hex.sb2

Feature Idea # 4

Include all 4 bases: decimal, binary, octal, and hexadecimal.

Design:

Now that we know how to display decimal and hexadecimal, adding binary and octal should be straightforward.

Since the sequence of operations when user clicks on any of the base names (Decimal, Binary, etc.) is pretty similar, we will create a generic custom procedure called 'show callback' which will take user input, convert it to all other bases and then display all numbers one by one.

```
Algorithm show callback:
Input: Base (2, 8, 10, or 16)
Get user input N in Base
Convert N to other bases
Call 'Show Number' 4 times to display each number
```

If you want to see how the program would look after all these changes, check out my program below:

File: baseconverter-allbases.sb2

Feature Idea # 5

Include START and STOP buttons to implement counting.

Design:

The idea of counting is simple. Counting starts at the current value and all bases start counting simultaneously. START starts counting, STOP stops it.

The "start" button will be connected to a script that will increment every number by 1 continuously. The "stop" button will stop this script. The coordination between start and stop can be achieved through the variable 'counter running'.

```
Algorithm Run counter:
Increment decimal by 1
Convert decimal to all other bases.
Call 'Show Number' 4 times to display each number
If counterRunning = True
      Call Run counter
End if
```

When user clicks on 'STOP' `counterRunning` will set to `False`, which will terminate the above recursion.

Feature Idea # 6

Include help and error detection.

Design:

Step 1: Include help.

Including help is straightforward. Clicking on "Help" button will bring up another sprite that will cover the entire screen and provide all require instructions. Clicking on it will make it go away.

Step 2: Verify user input.

The user may use incorrect digits, such as, 2001 for a binary number. We need to detect such errors because the subsequent conversion will go haywire.

Here is an algorithm for validating a number of a given base. We will use the custom block 'index' to get the location of a letter in the 'digits' array.

```
Input: nstr (number in string format), base (2, 8, 10, or 16)
for every letter c in nstr:
    position = index(c)
    if (position >= base)
        Number is invalid
    End if
End for
Number is valid
```

Save as Program Version “Final”

Congratulations! You have completed all the features of the program. As before, let’s save this project under a different name.

Compare your program with my program below.

File: baseconverter-final.sb2
Link: https://scratch.mit.edu/projects/549406885/

How to run the program:

1. Click Green flag. Click ‘help’ to view instructions.
2. Click any of the base names to enter a number of that base. The program then automatically shows that number in all 4 bases.
3. Click 'start' to start counting. The program starts counting from the current number.
4. Click 'stop' to stop counting.

Author: Abhay B. Joshi (abjoshi@yahoo.com)
Last modified: 29 June 2021