True or False

Morning

- Abstraction is an essential concept in Computer Science because every algorithm can be solved using abstraction to define patterns.

  FALSE: Abstraction does not solve algorithms

- Abstraction is an essential quality of an algorithm. Without it, an algorithm cannot be considered as an algorithm.

  FALSE: An algorithm needs to be unambiguous, executable, and terminating. You could write an algorithm that teaches someone how to get to UBC from your front door. The algorithm wouldn't have abstraction because it wouldn't work for most of the students at UBC but it would still be an algorithm.

- The best way to avoid bias in algorithms is to ensure that the data is recent.

  FALSE: Whether data is recent or not has no effect on bias. For example, if we took neighbourhood data from 2020, it doesn't mean that there are no biases because neighbourhood data inherently contains racial information. Neighbourhood data from 2020 wouldn't be any better than neighbourhood data from 1990; the same types of problems exist in both datasets.

- The best way to avoid bias in algorithms is to exclude racial information.

  FALSE: As discussed in class, sensitive info should be included to help decrease bias in data.

- Decomposition is the process of deciding how many pieces a problem needs to be broken down into. Typically, you need at least three smaller problems for decomposition to be beneficial.

  FALSE: Decomposition is not strictly about the number of problems, and nowhere did we specify that you need at least three smaller problems

- The best way to determine an algorithm's effectiveness is to determine how much space it takes on your computer.

  FALSE: The effectiveness of an algorithm is dependent on how accurate it is and whether it can do what it is claiming to be able to do. Space doesn't really play a role in the effectiveness (but it can play a role in efficiency).

- An algorithm should have an accuracy of 100% to be used in practice.

  FALSE: Sometimes, you don't know the full accuracy of an algorithm. For example, we build decision trees based on some training/test data. Will the decision tree be 100% accurate, we don't know but if we have done our jobs properly, it should be close. It doesn't mean we can't use it.

  Algorithms are used in the real world a lot even when they don't have 100% accuracy (e.g., facial recognition algorithms have high rates of error and are still used).

- The training data and the test data must be of equal percentages to ensure accuracy

  FALSE: This is not a rule and certainly isn't the only determining factor in determining accuracy. If you use bad data, even if you split the data 50/50 for test and training data, the accuracy of the resulting algorithm would still be low.

- When a computer adds two values together, the numbers are stored in the cache

  FALSE: The numbers must be stored in the registers

- The application displays a Say message every time a broadcast happens

  FALSE: There needs to be a snap block for what to do when a particular broadcast is heard.

- Imagine we asked the user to enter a number in Snap! (just like MED 4). A for-loop is the only way we can access each digit in the answer.

  FALSE: It is actually the "letter _ of _" block that allows you to access a particular digit or character in the answer.

- In countries where the writing system goes left to right and then top to bottom (e.g., Canada and English), using RLE is more beneficial since information about the images are also stored horizontally. This will help us save space compared to countries with other types of writing systems.

  FALSE: There is no correlation between RLE and the which writing system a country uses.

Afternoon

- To have an algorithm means that you already have abstraction.

  FALSE: An algorithm needs to be unambiguous, executable, and terminating. You could write an algorithm that teaches someone how to get to UBC from your front door. The algorithm wouldn't have abstraction because it wouldn't work for most of the students at UBC but it would still be an algorithm.

- The more abstraction an algorithm has, the more ambiguous it will be, so it will be more efficient.

  FALSE: Abstraction and efficiency aren't related in this way. Efficiency comes about from how your algorithm handles data (e.g., think about the difference in how selection sort handles data vs. simple sort). Just because an algorithm is more efficient, it doesn't necessarily mean that it is abstract.

  Consider the following algorithm to sort five cards:

  1. Swap the first and third card.
  2. Swap the second and third card.
  3. Swap the fourth and fifth card.

  You would have sorted all five cards in a more efficient way than selection sort but does this mean that this algorithm is more abstract?

- The best way to avoid bias in algorithms is to ensure that humans don't have any input.

  FALSE: Biases can be encoded into data by other non-human factors. For example, socioeconomic factors can cause biases in data.

- The best way to avoid bias in algorithms is to exclude location information.

  FALSE: Biases can be encoded into data by other non-location factors. For example, socioeconomic factors can cause biases in data.

- Decomposition is the process by which one takes a problem and breaks it down into smaller problems that are unsolvable.

  FALSE: It is about breaking it down into smaller problems **that are solvable.**

-

- Decomposition and abstraction always go hand in hand. You cannot have one without the other.

  FALSE: Decomposition and abstraction refer to two different things. Abstraction is the ability to generalize a situation (e.g., instead of being able to sort your exact shuffle order of cards, how can we generalize the situation to sort any shuffle order)? Decomposition is the process of breaking down a problem into smaller solvable problems (e.g., instead of sorting all 52 cards, let's sort 13 first).

- The best algorithms are always the fastest ones. When choosing between algorithms, you should always pick the one that takes the least amount of time.

  FALSE: It depends on what your needs are. If you are working on an older machine, it could be that you are constrained by the amount of memory available so you would choose an algorithm that is slower but uses less space.

- As long as there are attributes that do not appear in a decision tree, we need to continue calculating the entropy to determine the best decision tree.

  FALSE: If you can determine the result already, you do not need to include other attributes in your decision tree.

- The test data must be used to create the algorithm to avoid bias.

  FALSE: Test data, as the name implies, is used to test the algorithm you come up with from the training data. It will help you determine how accurate your algorithm is. It does not say anything about whether or not the algorithm is biased. If your data was biased in the first place, then any algorithm you create from it will be biased too. Garbage in, garbage out.

- A classifier that correctly predicts the result of its test data is considered unbiased.

  FALSE: If your data was biased in the first place, then any algorithm you create from it will be biased too. Garbage in, garbage out.

- Solid-state drives are critical because they are very big, and the data in them are very fast to access when compared to other forms of computer memory.

  FALSE: Yes, they are big, but accessing the data is very slow when compared to other forms of computer memory like registers or the cache.

-

- For a sprite to not react to a broadcast message, you need to include a stop block.

  FALSE: A sprite only reacts to a broadcast message if it includes a block that specifically says it should react to that message. Just because there are messages, does not mean a sprite will react to it. Using stop is typically reserved for repetitive structures.

- It is not possible for us to rewrite Snap! code such that all repeat until loops are replaced.

  FALSE: You can rewrite a repeat-until loop using other methods. For example, you could drag an operator (green) block into a repeat loop to replicate repeat-until.

- RLE uses an unnecessary number of bytes. A good improvement to the algorithm would be only to use one hexadecimal digit to represent runs of less than 16 pixels.

  FALSE: How would someone be able to tell whether a hexadecimal digit should be part of a colour or if it should be indicating the number of pixels in a run. For example, 342300000023345 ... We know the size is 34x23 and the first colour is white but is there a 2 pixel long run of white pixels or a 23 pixel long run? If we sometimes use one hexadecimal digit to represent the number of runs and at other times use two, it would be very hard to decode.

Sorting

Questions

- Assume you have 20 boxes already sorted in ascending order by height. Now, you want to sort the 20 boxes by weight. Thinking back to your CPSC 100 class, you remember simple, selection, and insertion sort; which sort would be the best to use in this situation? Why?

  Insertion sort is the best sort to use here because on average, insertion sort will take fewer comparisons when compared to selection sort. It is also definitely faster than simple sort. Selection and insertion sort take the same amount of space (which is less than simple sort).

  If you assumed that height and weight were correlated and thus, the boxes were already sorted, then either selection sort or insertion sort was an accepted answer provided there was a reasonable explanation.

- Assume you have 20 boxes already sorted in ascending order by weight. Now, you want to sort the 20 boxes by height. Thinking back to your CPSC 100 class, you remember simple, selection, and insertion sort; which sort would be the best to use in this situation? Why?

  Insertion sort is the best sort to use here because on average, insertion sort will take fewer comparisons when compared to selection sort. It is also definitely faster than simple sort. Selection and insertion sort take the same amount of space (which is less than simple sort).

  If you assumed that height and weight were correlated and thus, the boxes were already sorted, then either selection sort or insertion sort was an accepted answer provided there was a reasonable explanation.

- You are working as an admissions officer, and part of your job is to file paperwork. You have a cabinet drawer of applications, all neatly sorted by the last name. A new pile of applications has just come in, and you need to file each application. Thinking back to your CPSC 100 class, you remember simple, selection, and insertion sort; which sort would be the best to use in this situation? Why?

  Insertion sort because it is already pre-sorted.

-

- You have been assigned the task of sorting boxes and boxes of old papers chronologically (i.e., by time). You cannot scan or shred any of these papers. The room is super dusty and hot so you want to finish the task as fast as you can. Thinking back to your CPSC 100 class, you remember simple, selection, and insertion sort, which sort would be the best to use in this situation? Why?

  Insertion sort is the best to use because on average, it is faster than selection sort. Why?

  Selection sort and insertion sort are both sorts that have a worst case of $O(n^2)$. The n here represents the number of items you want to sort but the whole idea of $O(n^2)$ does not mean that you just square the number of items to get the total number of comparisons. The $O(n^2)$ indicates that the number of comparisons (or the time if you think about each comparison taking up a unit of time) required to complete the sort is generally double the number of items.

  On average, insertion sort is going to be a better choice than selection sort. Like you pointed out, the number of comparisons required by selection sort is always going to be the same for a given number of cards. However, for insertion sort, the number of comparisons required are going to depend on the order of your cards. If we have our cards in the worst possible order (e.g., largest to smallest when we want to sort them from smallest to largest), insertion sort and selection sort will both be $O(n^2)$.

  Imagine if you took your cards, shuffled them, and lay them on the table to sort. Now, let's say we do this multiple times. The likelihood of shuffling and having your cards in the worst possible order each time is exceptionally low. It's more likely that you'll end up with your cards in some order where one or more cards do not need to be swapped to the leftmost portion of the sorted list each time. On average, insertion sort is a better algorithm to use than selection sort.

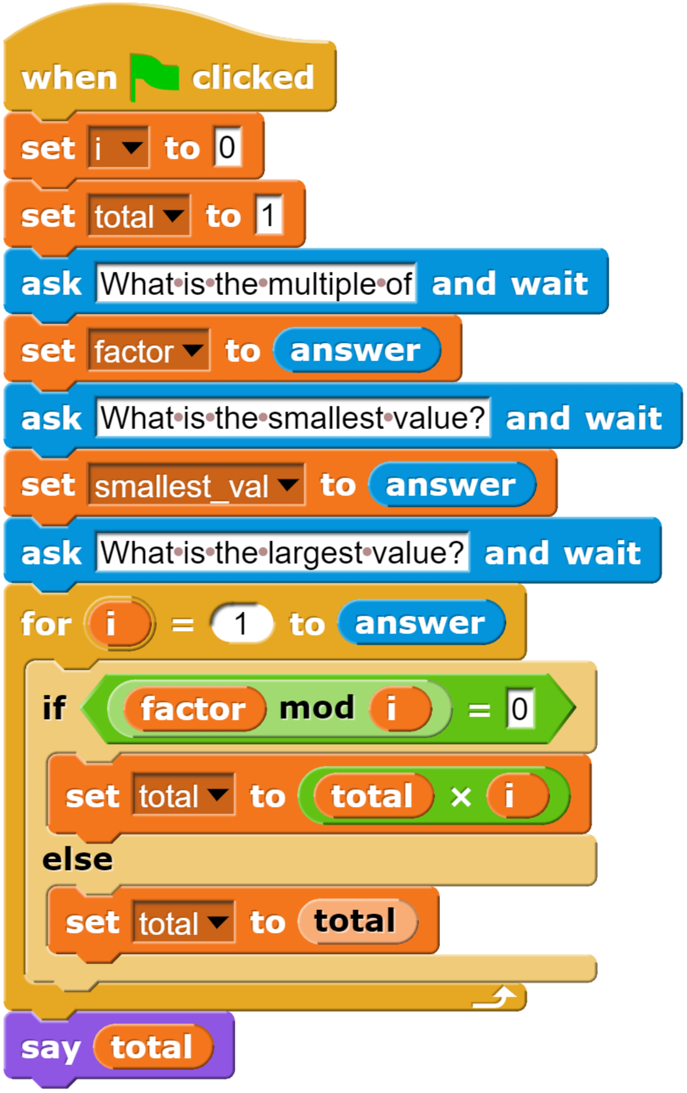
Bugaboo

The block depicted is supposed to find the product between two positive integers. For instance, if the user enters 2, 1, and 10, the result should be 2*4*6*8 (i.e., you obtain the product from multiplying all the multiples of 2 between 1 and 10).

Take a look at the code below and determine if there are any bugs. If the code is correct, be sure to state that the code is correct (blank answers are considered as no answer). If the code is incorrect, identify any bugs and, in English, describe what needs to be changed so that the code works properly.

Morning

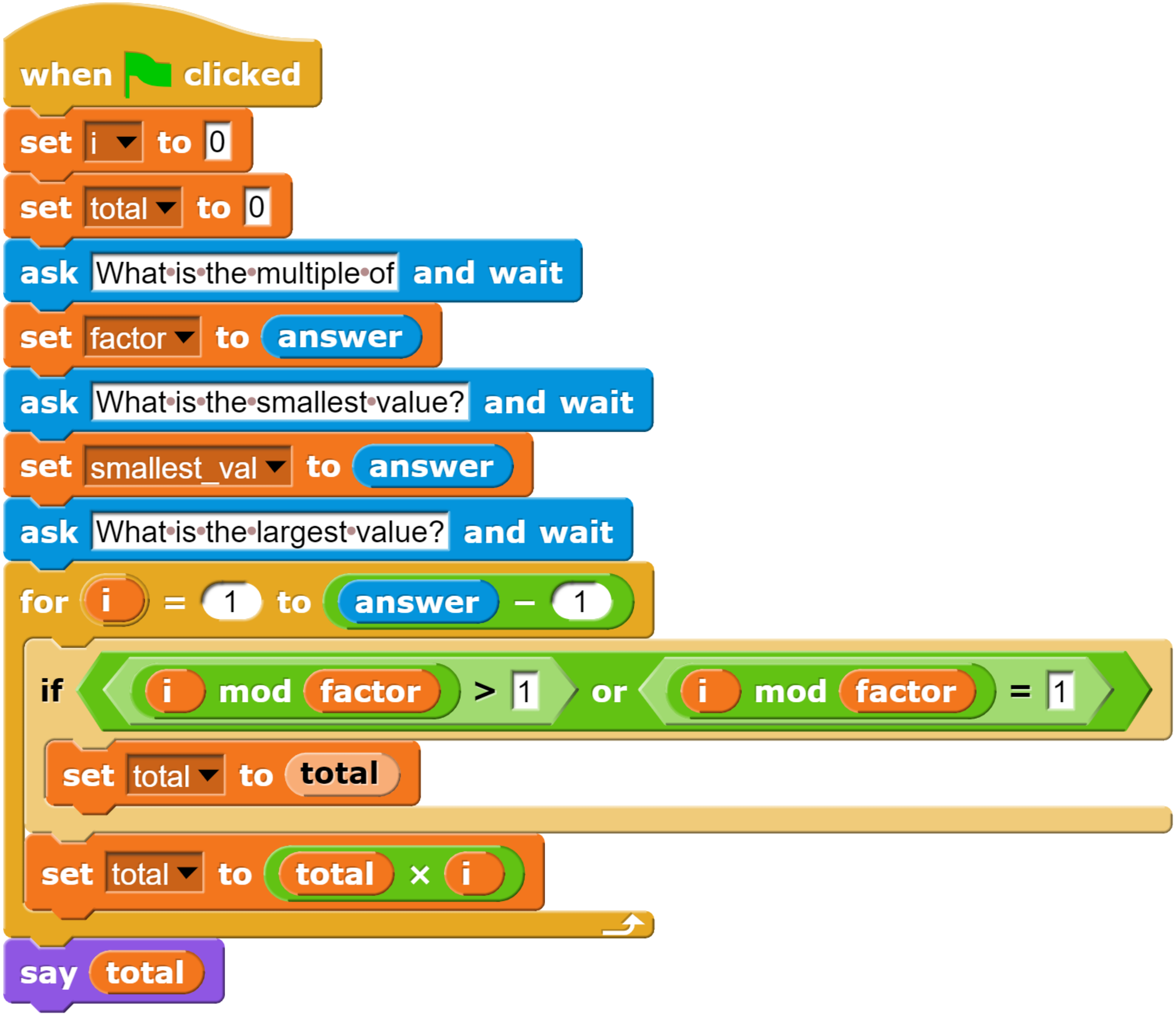
Version 1



```
when ⚑ clicked
set i to 0
set total to 1
ask What is the multiple of and wait
set factor to answer
ask What is the smallest value? and wait
set smallest_val to answer
ask What is the largest value? and wait
for i = 1 to answer
    if (factor mod i) = 0
        set total to (total × i)
    else
        set total to total
say total
```

- The for loop should start from i = smallest_val+1 or smallest_val is not used
- The for loop should end at answer-1
- The if-statement inside the for loop should be i mod factor and not factor mod i

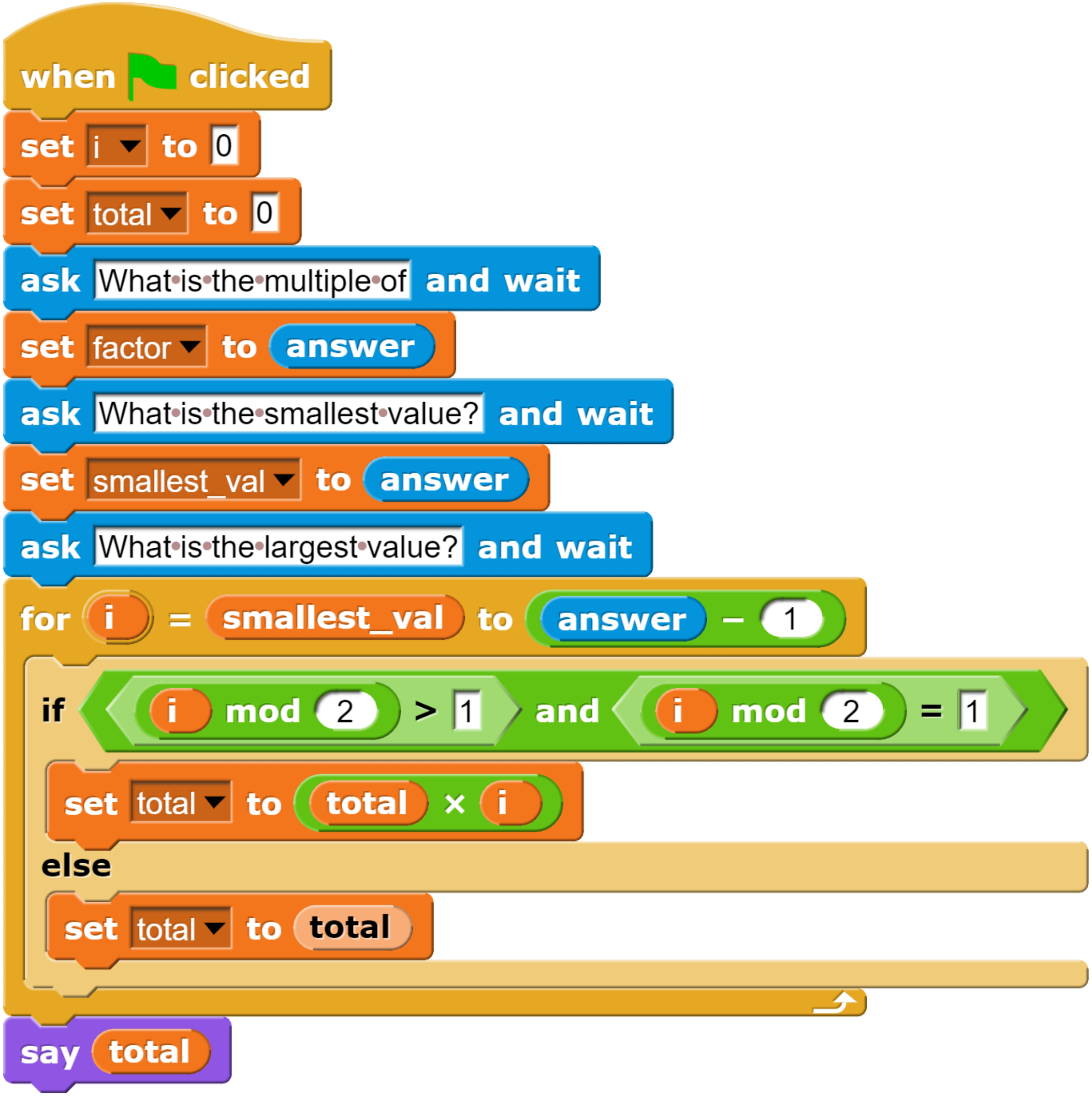
Version 2



```
when ⚑ clicked
set i to 0
set total to 0
ask What is the multiple of and wait
set factor to answer
ask What is the smallest value? and wait
set smallest_val to answer
ask What is the largest value? and wait
for i = 1 to answer − 1
    if i mod factor > 1 or i mod factor = 1
        set total to total
    set total to total × i
say total
```

- The for loop should start from i = smallest_val+1 OR stores smallest_value but doesn't use it
- Total should be initialized to 1, not 0
- "Set total to total x i block" should be inside an else block

Version 3



```
when ⚑ clicked
set i to 0
set total to 0
ask What is the multiple of and wait
set factor to answer
ask What is the smallest value? and wait
set smallest_val to answer
ask What is the largest value? and wait
for i = smallest_val to answer − 1
    if ((i mod 2) > 1) and ((i mod 2) = 1)
        set total to total × i
    else
        set total to total
say total
```

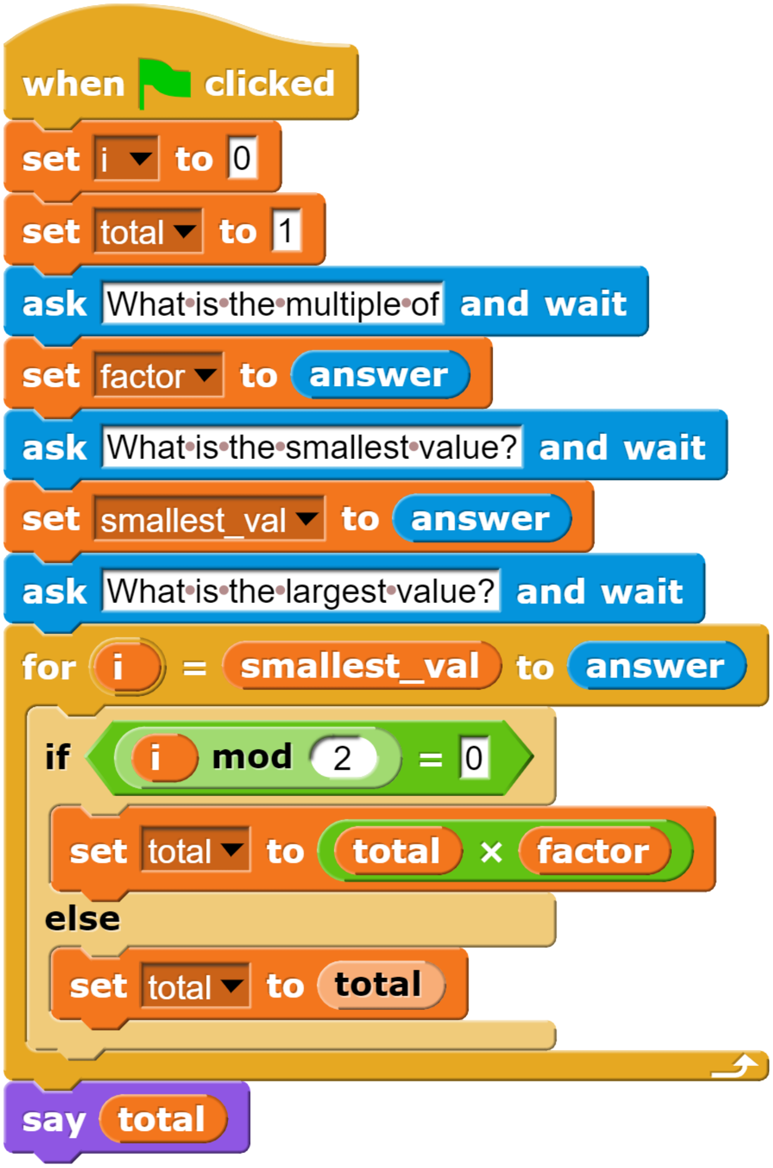
- The condition listed in the if-statement is incorrect
    - It should not be mod 2
    - There shouldn’t be an AND condition. There is no way for a number to be both greater than and equal to 1.

- Total should be initialized to 1, not 0

- The loop should start at smallest_val+1

Afternoon

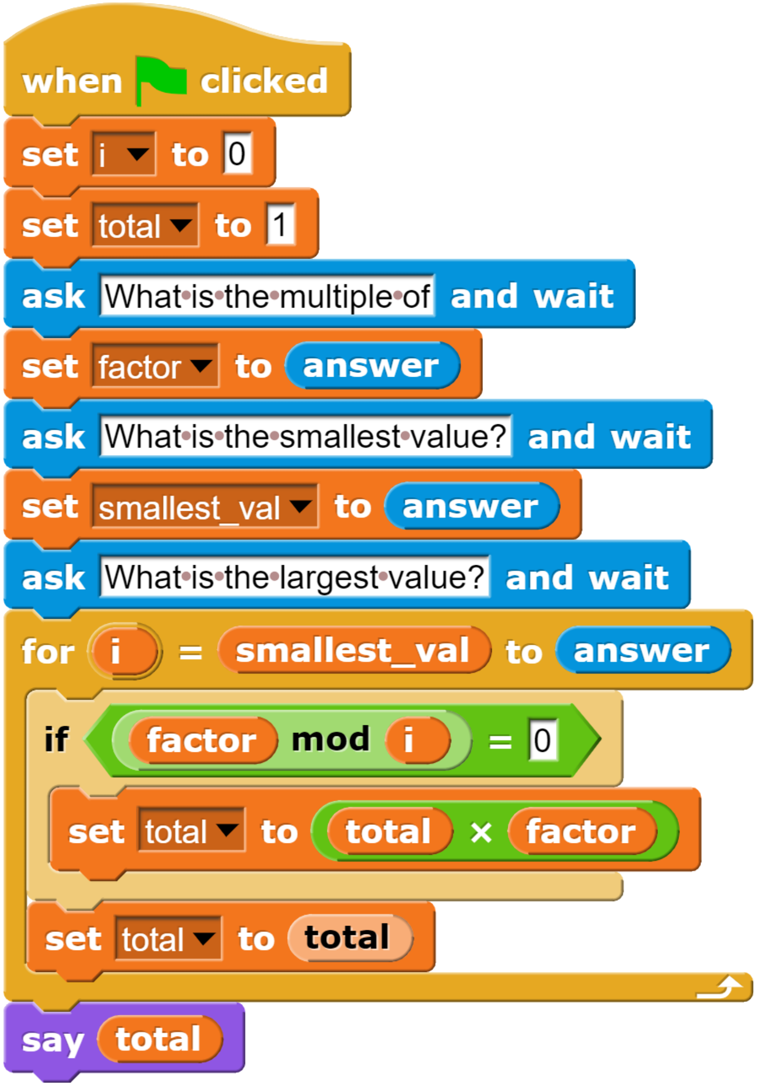
Version 1



```
when ⚑ clicked
set i to 0
set total to 1
ask What is the multiple of and wait
set factor to answer
ask What is the smallest value? and wait
set smallest_val to answer
ask What is the largest value? and wait
for i = smallest_val to answer
    if ((i mod 2) = 0)
        set total to (total × factor)
    else
        set total to total
say total
```

- The for loop should end at answer-1, not answer OR we should be starting at smallest_val + 1
- Inside the if statement, total should be set to total x i when the condition is true
- The if condition itself is wrong. It should not be i mod 2, it should be i mod factor

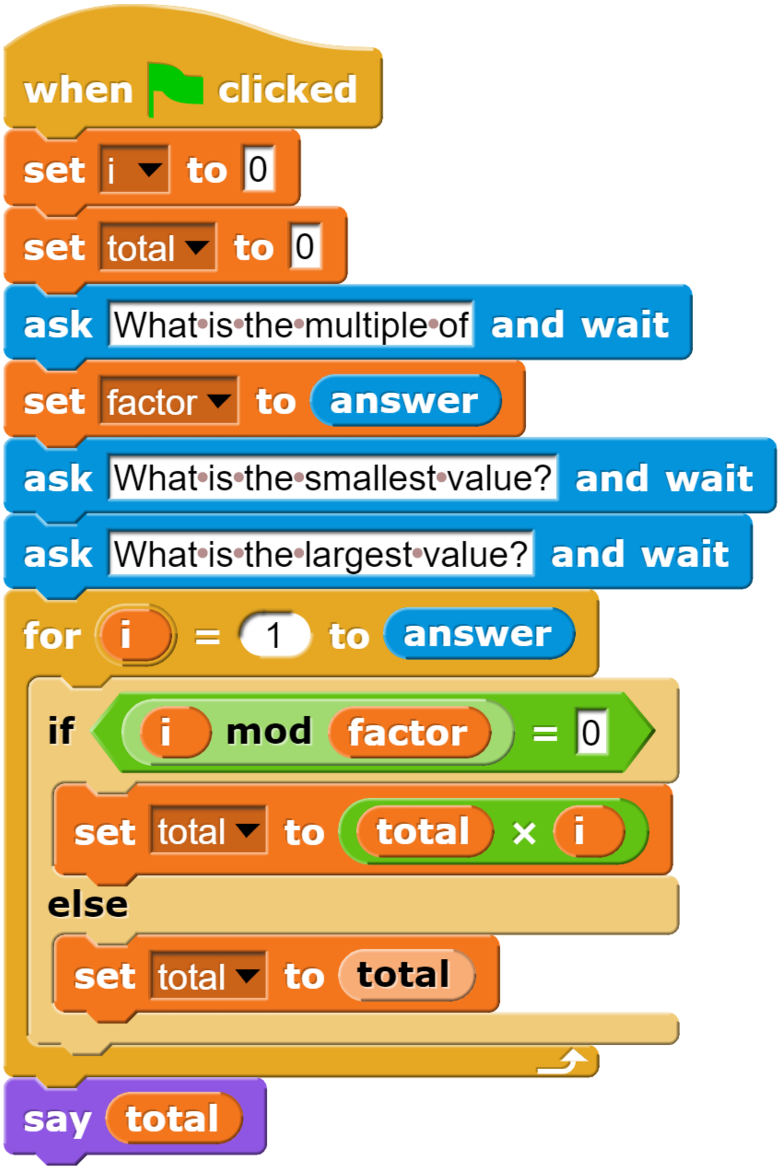
Version 2



```
when 🏁 clicked
set i to 0
set total to 1
ask What·is·the·multiple·of and wait
set factor to answer
ask What·is·the·smallest·value? and wait
set smallest_val to answer
ask What·is·the·largest·value? and wait
for i = smallest_val to answer
    if ((factor mod i) = 0)
        set total to (total × factor)
    set total to total
say total
```

- The for-loop should end at answer-1, not answer OR the for-loop should start at smallest_val +1
- The if condition should say i mod factor, not factor mod i
- Total should be total x i not total x factor

Version 3



```
when [flag] clicked
set i to 0
set total to 0
ask What is the multiple of and wait
set factor to answer
ask What is the smallest value? and wait
ask What is the largest value? and wait
for i = 1 to answer
    if (i mod factor) = 0
        set total to total × i
    else
        set total to total
say total
```

- The for loop should end at answer-1, not answer
- The smallest value is not stored. You always start at 1 regardless of what the smallest value is.
- Total should not be initialized to 0

Data Representation (Numbers)

Morning

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 6 symbols, how many symbols can we represent with 8 digits?

  Answer: 1679616

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 7 symbols, how many symbols can we represent with 3 digits?

  Answer: 343

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 4 symbols, how many symbols can we represent with 7 digits?

  Answer: 16384

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 4 symbols, how many digits would we need to represent 100 symbols?

  Answer: 4

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 3 symbols, how many digits would we need to represent 200 symbols?

  Answer: 5

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 9 symbols, how many digits would we need to represent 250 symbols?

  Answer: 3

Afternoon

- Let’s say that instead of having a base 2 number system, we have a new type of system where each digit represents 3 symbols, how many symbols can we represent with 8 digits?

  Answer: 6561

- Let’s say that instead of having a base 2 number system, we have a new type of system where each digit represents 9 symbols, how many symbols can we represent with 3 digits?

  Answer: 729

- Let’s say that instead of having a base 2 number system, we have a new type of system where each digit represents 5 symbols, how many symbols can we represent with 6 digits?

  Answer: 15625

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 11 symbols, how many digits would we need to represent 1000 symbols?

  Answer: 3

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 9 symbols, how many digits would we need to represent 500 symbols?

  Answer: 3

- Let’s say that instead of having a base 10 number system, we have a new type of system where each digit represents 7 symbols, how many digits would we need to represent 400 symbols?

  Answer: 4

Image Representation

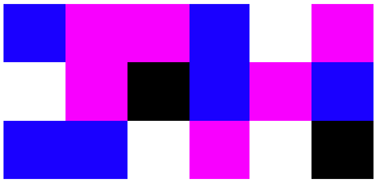
Morning

You are working for an Image Processing and Printing company that has designed a new bitmap specification. They are also in the process of trying out new encoding formats.

The new bitmap specification you are given is:

- The first byte indicates the image width
- The second byte indicates image height
- Each pixel's RGB values (3 bytes each) are stored. The order in which we store the pixel data is by reading from left to right and then from top to bottom.

Version 1



How many bytes would it take to represent the above image?

$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 18\ pixels\ =\ 56\ bytes$

If we were to use RLE with this new specification, how many bytes would it take to represent the image?

The RLE for this image would be:
06 03
0000FF 01 FF00FF 02 0000FF 01 FFFFFF 01 FF00FF 01
FFFFFF 01 FF00FF 01 000000 01 0000FF 01 FF00FF 01 0000FF 03
FFFFFF 01 FF00FF 01 FFFFFF 01 000000 01

$2\ bytes\ +\ \frac{4\ bytes}{run} \times 15\ runs\ =\ 62\ bytes$

What are the % savings for this image when we use RLE?

You would not save by using RLE.

4. Imagine that we have developed a new compression encoding called ENCODE. Here is the algorithm for creating a representation in ENCODE:
    1. Create a table with two columns. One column is called “Lookup Index” and the other is called “Colour”.
        1. All values stored in this table will be hexadecimal digits.
        2. You can safely assume that the maximum number of different colours in an image is 255.

    2. For each pixel in the image, get the colour. Check if the colour is already listed in the table.
        1. If it is, move to the next pixel.
        2. If not, create an entry in the table for this colour. For example, if the image contained the colour white, the table could have an entry with lookup index = 00 and colour = FFFFFF. If we then found the colour blue, the table would then contain lookup index = 01 and colour = 0000FF.

    3. Repeat step b for all the pixels in the image.

    4. Use RLE to create your representation only instead of using the colour representation in the encoding, use the lookup index

    What would be the compressed representation of the image using ENCODE and the specification given above?

    NOTE:

    Because colors are displayed differently on computers, we do not want you to **spend much time trying to find the "right shade of any color".** Please do not spend a lot of time trying to decide on what hex code to use. So for instance, for red you can use FF0000 you don't need to find the right shade so trying to do FF1209 is a waste of time. We recommend that you use max (FF), min (00) and average (80) values for each color.

    So for instance:

    - Red would be FF0000
    - Grey would be 808080
    - Yellow would be FFFF00

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	Colour
00	0000FF
01	FF00FF
02	FFFFFF
03	000000

The representation derived from using ENCODE to represent the image above is:
06 03
00 01 01 02 00 01 02 01 01 01
02 01 01 01 03 01 00 01 01 01 00 03
02 01 01 01 02 01 03 01

5. How many bytes would be needed to represent the image if we were to use ENCODE?

Without the Table

The total number of bytes required is: $2\ bytes\ +\ \frac{2\ bytes}{run} \times 15\ run\ =\ 32\ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

With the Table

The total number of bytes including the table is:

$2\ bytes\ +\ \frac{2\ bytes}{run} \times 15\ run\ +\ 16\ bytes\ =\ 48\ bytes$

The additional 16 bytes refers to the bytes required for the table (4 bytes = value column, 12 bytes = colour column)

6. What would be the % savings?

Without the Table

ENCODE Against Uncompressed Images

$[(56\ bytes\ -\ 32\ bytes)\ \div\ 56\ bytes]\ \times\ 100\%\ =\ 42.8\%$

You could also round this to 43%.

ENCODE Against RLE

$[(62\,bytes\ -\ 32\,bytes)\ \div\ 62\,bytes]\ \times\ 100\%\ =\ 48.4\%$

You could also round this to 48%.

With the Table

ENCODE Against Uncompressed Images

$[(56\,bytes\ -\ 48\,bytes)\ \div\ 56\,bytes]\ \times\ 100\%\ =\ 14.3\%$

You could also round this to 14%.

ENCODE Against RLE

$[(62\,bytes\ -\ 48\,bytes)\ \div\ 62\,bytes]\ \times\ 100\%\ =\ 22.6\%$

You could also round this to 23%.

7. Describe a situation in which using ENCODE would be better than using RLE?

   Assuming there are no more than 16777215 (the decimal value of FFFFFF) colours in an image, using ENCODE is always better because our “savings” come from the fact that a colour can now be represented by less than three bytes.

8. Describe a situation in which using ENCODE would be worse than using RLE?

   If there were more than 16777215 colours in an image, then the lookup value for a colour would start to be bigger than three bytes. In that case, using RLE would be better.

9. Is ENCODE an example of a lossy or lossless compression format? Why or why not?

   ENCODE is a lossless compression format because we don’t lose any data. We can easily recreate the originalargest numl image by using the lookup table to get the exact value of the pixels.

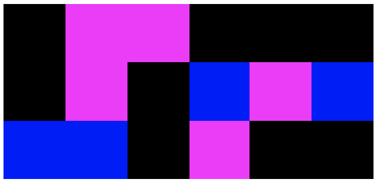
10. What is the largest number of pixels that we can represent with this specification?

    Since width and height are both represented as two hexadecimal digits, that means the largest decimal value for width and height would be 255 (0xFF), If both width and height are 255 pixels large, then the largest number of pixels we can represent is 65025.

11. If we wanted to handle a larger number of pixels, what would we need to change in the specification?

    We would need more bytes allocated to the width and height.

Version 2



How many bytes would it take to represent the above image?

$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 18\ pixels\ =\ 56\ bytes$

12. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

The RLE for this image would be:
06 03
000000 01 FF00FF 02 000000 04
FF00FF 01 000000 01 0000FF 01 FF00FF 01 0000FF 03
000000 01 FF00FF 01 000000 02

$2\ bytes\ +\ \frac{4\ bytes}{run} \times\ \ 11\ runs\ =\ 46\ bytes$

13. What are the % savings for this image when we use RLE?

(56 - 46) / 56 * 100 = 17.9 = 18% Savings

14. What would the ENCODE representation be?

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	**Colour**
00	000000
01	FF00FF
02	0000FF
03	FFFFFF

The representation derived from using ENCODE to represent the image above is:
06 03
00 01 **01** 02 **00** 04
01 01 **00** 01 **02** 01 **01** 01 **02** 03
00 01 **01** 01 **00** 02

15. How many bytes would be needed to represent the image if we were to use ENCODE?

The total number of bytes required is: $2\ bytes + \frac{2\ bytes}{run} \times 11\ runs = 24\ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

16. What would be the % savings?

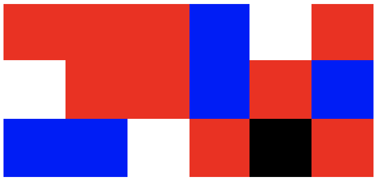
ENCODE Against Uncompressed Images
$[(56\ bytes - 24\ bytes) \div 56\ bytes] \times 100\% = 57.1\%$

You could also round this to 57%.

ENCODE Against RLE
$[(46\ bytes - 24\ bytes) \div 46\ bytes] \times 100\% = 47.8\%$

You could also round this to 48%.

Version 3



How many bytes would it take to represent the above image?

$$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 18\ pixels\ =\ 56\ bytes$$

17. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

The RLE for this image would be:
06 03

FF0000 03 0000FF 01 FFFFFF 01 FF0000 01
FFFFFF 01 FF0000 02 0000FF 01 FF0000 01 0000FF 03
FFFFFF 01 FF0000 01 000000 01 FF0000 01

$$2\ bytes\ +\ \frac{4\ bytes}{run} \times\ \ 13\ \ runs\ =\ 54\ bytes$$

18. What are the % savings for this image when we use RLE?

(56-54) / 56 * 100 = 3.57

4% savings

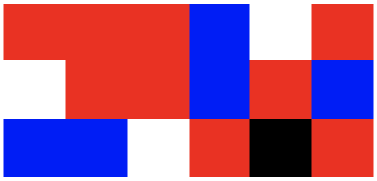
19. What would the ENCODE representation be?

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	**Colour**
00	FF0000
01	0000FF

02	FFFFFF
03	000000

The representation derived from using ENCODE to represent the image above is:
06 03
00 **03** 01 **01** 02 **01** 00 **01**
02 **01** 00 **02** 01 **01** 00 **01** 01 **03**
02 **01** 00 **01** 03 **01** 00 **01**



20. How many bytes would be needed to represent the image if we were to use ENCODE?

The total number of bytes required is: $2\,bytes\ +\ \frac{2\,bytes}{run} \times 13\,runs\ =\ 28\,bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

21. What would be the % savings?

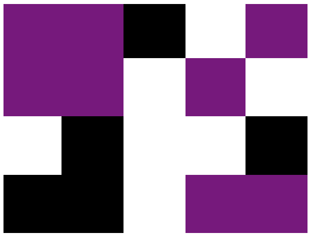
ENCODE Against Uncompressed Images
$[(56\,bytes\ -\ 28\,bytes)\ \div\ 56\,bytes]\ \times\ 100\%\ =\ 50.0\%$

You could also round this to 50%.

ENCODE Against RLE
$[(54\,bytes\ -\ 28\,bytes)\ \div\ 54\,bytes]\ \times\ 100\%\ =\ 48.1\%$

You could also round this to 48%.

Version 4 - 27 / 37-41



How many bytes would it take to represent the above image?

$$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 20\ pixels\ =\ 62\ bytes$$

22. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

The RLE for this image would be:
05 04
FF00FF 02 000000 01 FFFFFF 01 FF00FF 03
FFFFFF 01 FF00FF 01 FFFFFF 02
000000 01 FFFFFF 02 000000 03
FFFFFF 01 FF00FF 02

$$2\ bytes\ +\ \frac{4\ bytes}{run}\ \times\ \ 12\ \ runs\ =\ 50\ bytes$$

23. What are the % savings for this image when we use RLE?
(62 bytes - 50 bytes) / 62 bytes X 100% = 19.35%

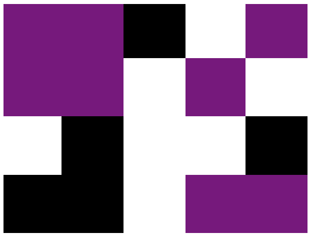
You could also round this to 19%.

24. What would the ENCODE representation be?

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	Colour
00	FF00FF
01	000000
02	FFFFFF

The representation derived from using ENCODE to represent the image above is:



05 04
00 02 **01** 01 02 01 **00** 03
02 01 00 01 02 02
01 01 02 02 01 03
02 01 00 02

25. How many bytes would be needed to represent the image if we were to use ENCODE?

The total number of bytes required is: $2\ bytes\ +\ \frac{2\ bytes}{run} \times 12\ runs\ =\ 26\ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

26. What would be the % savings?

Without the table

ENCODE Against Uncompressed Images

$$[(62\ bytes\ -\ 26\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 58.1\%$$

You could also round this to 58%.

ENCODE Against RLE

$$[(50\ bytes\ -\ 26\ bytes)\ \div\ 50\ bytes]\ \times\ 100\%\ =\ 48.0\%$$

You could also round this to 48%.

With the table
(26 bytes + 12 bytes for the table = 38 bytes)
ENCODE Against Uncompressed Images

$$[(62\ bytes\ -\ 38\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 38.7\%$$

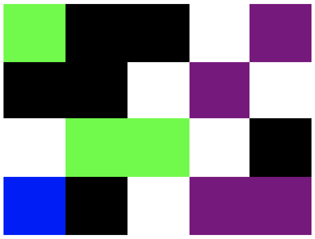
You could also round this to 39%.

ENCODE Against RLE

$$[(50\ bytes\ -\ 38\ bytes)\ \div\ 50\ bytes]\ \times\ 100\%\ =\ 24.0\%$$

You could also round this to 24%.

Version 5



How many bytes would it take to represent the above image?

$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 20\ pixels\ =\ 62\ bytes$

27. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

    The RLE for this image would be:
    05 04
    00FF00 01 000000 02 FFFFFF 01 FF00FF 01
    000000 02 FFFFFF 01 FF00FF 01 FFFFFF 02
    00FF00 02 FFFFFF 01 000000 01
    0000FF 01 000000 01 FFFFFF 01 FF00FF 02

    $2\ bytes\ +\ \frac{4\ bytes}{run} \times\ \ 15\ \ runs\ =\ 62\ bytes$

28. What are the % savings for this image when we use RLE?

    You would not save by using RLE.

29. What would the ENCODE representation be?

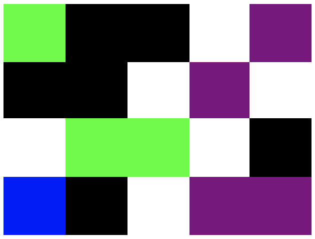
    The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	Colour
00	00FF00
01	000000
02	FFFFFF

03	FF00FF
04	0000FF

The representation derived from using ENCODE to represent the image above is:

05 04
00 **01** 01 **02** 02 **01** 03 **01**
01 **02** 02 **01** 03 **01** 02 02
00 **02** 02 **01** 01 **01**
04 **01** 01 **01** 02 **01** 03 **02**



30. How many bytes would be needed to represent the image if we were to use ENCODE?

The total number of bytes required is: $2\ bytes\ +\ \frac{2\ bytes}{run} \times 15\ runs\ =\ 32\ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

31. What would be the % savings?

Without the table

ENCODE Against Uncompressed Images

$[(62\ bytes\ -\ 32\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 48.4\%$

You could also round this to 48%.

ENCODE Against RLE

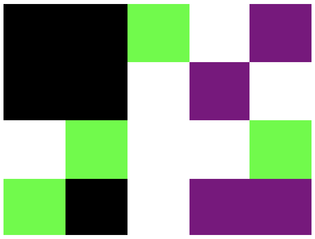
$[(62\ bytes\ -\ 32\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 48.4\%$

You could also round this to 48%.

With the table

You would not save by using ENCODE.

Version 6



How many bytes would it take to represent the above image?

$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 18\ pixels\ =\ 62\ bytes$

32. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

The RLE for this image would be:

05 04
000000 02 00FF00 01 FFFFFF 01 FF00FF 01
000000 02 FFFFFF 01 FF00FF 01 FFFFFF 02
00FF00 01 FFFFFF 02 00FF00 02
000000 01 FFFFFF 01 FF00FF 02
$2\ bytes\ +\ \frac{4\ bytes}{run} \times\ \ 14\ \ runs\ \ =58\ bytes$

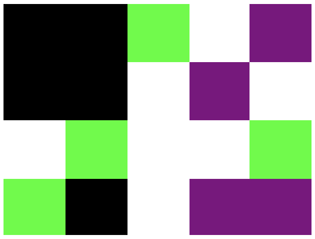
33. What are the % savings for this image when we use RLE?
(62 bytes - 58 bytes) / 62 bytes X 100% = 6.45%
You could also round this to 7%.
34. What would the ENCODE representation be?

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	**Colour**
00	000000
01	00FF00
02	FFFFFF

03	FF00FF

The representation derived from using ENCODE to represent the image above is:
05 04
00 **02** 01 **01** 02 **01** 03 01
00 **02** 02 **01** 03 **01** 02 **02**
01 **01** 02 **02** 01 **02**
00 **01** 02 **01** 03 **02**



35. How many bytes would be needed to represent the image if we were to use ENCODE?
Without the table

The total number of bytes required is: $2\ bytes\ +\ \frac{2\ bytes}{run} \times 14\ runs\ =\ 30\ \ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

With the table

The total number of bytes including the table is:
$2\ bytes\ +\ \frac{2\ bytes}{run} \times 14\ run\ +\ 16\ bytes\ =\ 46\ bytes$

The additional 16 bytes refers to the bytes required for the table (4 bytes = value column, 12 bytes = colour column)

36. What would be the % savings?
Without the table

ENCODE Against Uncompressed Images
$[(62\ bytes\ -\ 30\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 51.6\%$

You could also round this to 52%.

ENCODE Against RLE

$$[(58\, bytes\ -\ 30\, bytes)\ \div\ 58\, bytes]\ \times\ 100\%\ =\ 48.27\%$$

You could also round this to 48%.

With table

ENCODE Against Uncompressed Images

$$[(62\, bytes\ -\ 46\, bytes)\ \div\ 62\, bytes]\ \times\ 100\%\ =\ 25.8\%$$

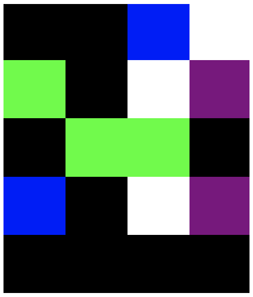
You could also round this to 26%.

ENCODE Against RLE

$$[(58\, bytes\ -\ 46\, bytes)\ \div\ 58\, bytes]\ \times\ 100\%\ =\ 20.68\%$$

You could also round this to 21%.

Version 7



How many bytes would it take to represent the above image?

$$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times\ 20\ pixels\ =\ 62\ bytes$$

37. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

The RLE for this image would be:
04 05
000000 02 0000FF 01 FFFFFF 01
00FF00 01 000000 01 FFFFFF 01 FF00FF 01
000000 01 00FF00 02 000000 01
0000FF 01 000000 01 FFFFFF 01 FF00FF 01
000000 04

$$2\ bytes\ +\ \frac{4\ bytes}{run} \times\ \ 15\ \ runs\ =\ 62\ bytes$$

38. What are the % savings for this image when we use RLE?

Bytes to rep pic: $2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 20\ pixels\ =\ 62\ bytes$

You would not save by using RLE

39. What would the ENCODE representation be?

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	Colour
00	000000
01	0000FF
02	FFFFFF
03	00FF00
04	FF00FF

The representation derived from using ENCODE to represent the image above is:
04 05

00 02 01 01 02 01
03 01 00 01 02 01 04 01
00 01 03 02 00 01
01 01 00 01 02 01 04 01
00 04

40. How many bytes would be needed to represent the image if we were to use ENCODE?
Without the table

The total number of bytes required is: $2\ bytes\ +\ \frac{2\ bytes}{run} \times 15\ runs\ =\ 32\ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

With the Table

The total number of bytes including the table is:

$2\ bytes\ +\ \frac{2\ bytes}{run} \times 15\ run\ +\ 20\ bytes\ =\ 52\ \ bytes$

The additional 20 bytes refers to the bytes required for the table (5 bytes = value column, 15 bytes = colour column)

41. What would be the % savings?

Without the table

ENCODE Against Uncompressed Images

$[(62\ bytes\ -\ 32\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 48.4\%$

You could also round this to 48%.

ENCODE Against RLE

$[(62\ bytes\ -\ 32\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 48.4\%$

With the table:

ENCODE Against Uncompressed Images

$[(62\ \ bytes\ -\ 52\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 16.1\%$

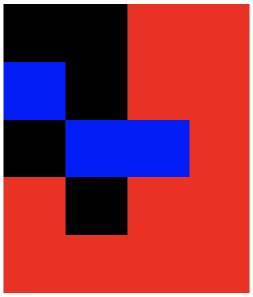
You could also round this to 16%.

ENCODE Against RLE

$[(62\ \ bytes\ -\ 52\ \ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 16.1\%$

You could also round this to 16%.

Version 8



How many bytes would it take to represent the above image?

$2\ bytes\ +\ \frac{3\ bytes}{pixel}\times\ 20\ pixels\ =\ 62\ bytes$

42. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

The RLE for this image would be:

$2\ bytes\ +\ \frac{4\ bytes}{run} \times\ \ 10\ \ runs\ \ =\ \ 42\ bytes$

04 05
000000 02 FF0000 02
0000FF 01 000000 01 FF0000 02
000000 01 0000FF 02 FF0000 02
000000 01 FF0000 06

43. What are the % savings for this image when we use RLE?

Uncompressed image bytes: $2\ bytes\ +\ \frac{3\ bytes}{pixel} \times\ 20\ pixels\ \ =\ \ 62\ bytes$

$[(62\ bytes\ \ -\ \ 42\ bytes)\ \ \div\ \ 62\ bytes]\ \ \times\ \ 100\%\ \ =\ \ 32.26\%$

You could also round this to 32%.

44. What would the ENCODE representation be?

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	**Colour**
00	000000
01	FF0000
02	0000FF

The representation derived from using ENCODE to represent the image above is:
04 05
00 02 01 02
02 01 00 01 01 02
00 01 02 02 01 02
00 01 01 06

45. How many bytes would be needed to represent the image if we were to use ENCODE?

With the Table:

The total number of bytes required is:
$12\ bytes\ +2\ bytes\ +\ \frac{2\ bytes}{run} \times 10\ \ runs\ =\ 34\ bytes$

Without the table:
The total number of bytes required is: $2\ bytes\ +\ \frac{2\ bytes}{run} \times 10\ \ runs\ =\ 22\ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

46. What would be the % savings?

ENCODE (with table) Against Uncompressed Images
$[(62\ bytes\ -\ 34\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 45.2\%$

You could also round this to 45%.

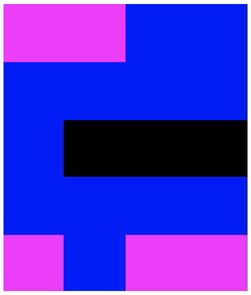
ENCODE (without table) Against Uncompressed Images
$[(62\ bytes\ -\ 22\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 64.5\%$

You could also round this to 65%.

ENCODE (witth table) Against RLE
$[(42\ bytes\ -\ 34\ bytes)\ \div\ 42\ bytes]\ \times\ 100\%\ =\ 19\%$

ENCODE (without table) Against RLE
$[(42\ bytes\ -\ 22\ bytes)\ \div\ 42\ bytes]\ \times\ 100\%\ =\ 47.62\%$

Version 9



How many bytes would it take to represent the above image?

$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 20\ \ pixels\ \ =\ \ 62\ bytes$

The RLE for this image would be:

$2\ bytes\ +\ \frac{4\ bytes}{run} \times\ \ \ 7\ \ runs\ \ =\ \ 30\ bytes$

04 05
FF80FF 02 0000FF 07
000000 03
0000FF 04
FF80FF 01 0000FF 01 FF80FF 02

47. What are the % savings for this image when we use RLE?

Uncompressed image bytes: $2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 20\ pixels\ \ =\ \ 62\ bytes$

$[(62\ bytes\ \ -\ \ 30\ bytes)\ \ \div\ \ 62\ bytes]\ \times\ 100\%\ \ =\ \ 51.6\%$

You can also round this to 52%

48. What would the ENCODE representation be?

The lookup table would look like this (remember, the lookup table is NOT part of your encoding):

Lookup Value	Colour
00	FF80FF

01	0000FF
02	000000

The representation derived from using ENCODE to represent the image above is:

04 05
00 02 01 07
02 03 01 04
00 01 01 01 00 02

49. How many bytes would be needed to represent the image if we were to use ENCODE?

Without the table:

The total number of bytes required is: $2\ bytes\ +\ \frac{2\ bytes}{run} \times 7\ runs\ =\ 16\ bytes$

With the table:

The total number of bytes required is:

$12\ bytes\ +\ 2\ bytes\ +\ \frac{2\ bytes}{run} \times 7\ runs\ =\ 28\ bytes$

Another way to double check your answer is to realize that a colour is now represented by 1 byte as opposed to 3 bytes. Therefore, each run now only requires 2 bytes instead of 4.

50. What would be the % savings?

ENCODE (without table) Against Uncompressed Images

$[(62\ bytes\ -\ 16\ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 74.2\ \%$

You could also round this to 74%.

ENCODE (with table) Against Uncompressed Images

$[(62\ bytes\ -\ 28\ \ bytes)\ \div\ 62\ bytes]\ \times\ 100\%\ =\ 54.8\%$

You could also round this to 55%.

ENCODE (without table) Against RLE

$[(30\ bytes\ -\ 16\ \ bytes)\ \div\ 30\ bytes]\ \times\ 100\%\ =\ 46.7\ \%$

You could also round this to 47%.

ENCODE (with table) Against RLE

$[(30\ bytes\ -\ 28\ bytes)\ \div\ 30\ bytes]\ \times\ 100\%\ =\ 6.7\%$

You could also round this to 7%.

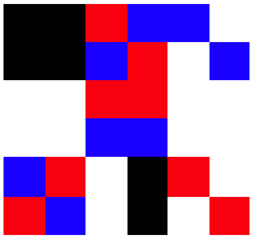
Afternoon

You are working for an Image Processing and Printing company that has designed a new bitmap specification. They are also in the process of trying out new encoding formats.

The new bitmap specification you are given is:

- The first byte indicates the image height
- The second byte indicates image width
- Each pixel's RGB values (3 bytes each) are stored. The order in which we store the pixel data is by reading from left to right and then from top to bottom.

Version 1



1. How many bytes would it take to represent the above image?

   $2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 36\ pixels\ =\ 110\ bytes$

2. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

   06 06
   000000 02 FF0000 01 0000FF 02 FFFFFF 01
   000000 02 0000FF 01 FF0000 01 FFFFFF 01 0000FF 01
   FFFFFF 02 FF0000 02 FFFFFF 04
   0000FF 02 FFFFFF 02
   0000FF 01 FF0000 01 FFFFFF 01 000000 01 FF0000 01 FFFFFF 01
   FF0000 01 0000FF 01 FFFFFF 01 000000 01 FFFFFF 01 FF0000 01

   $2\ bytes\ +\ \frac{4\ \ bytes}{run} \times 26\ runs\ =\ 106\ bytes$

3. What are the % savings for this image when we use RLE?

   $[(110\ bytes\ -\ 106\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 3.6\%$
   You can also round this to 4%

4. Imagine that we have developed a new compression encoding called ENCODE. Here is the algorithm for creating a representation in ENCODE:
    1. Starting from the top left, divide the pixels into 2x2 blocks.
        1. A pixel can only ever belong to a single block.
        2. You can safely assume that an image can be evenly divided into 2x2 blocks (i.e., there will always be an even number of rows and columns).

    2. Calculate the colour of each block by averaging all the pixels in the block.

    3. Use RLE to create your representation only instead of using the colour representation of each pixel, use the colour representation of the block.

    What would be the compressed representation of the image using ENCODE and the specification given above?

    NOTE:

    Because colors are displayed differently on computers, we do not want you to **spend much time trying to find the "right shade of any color".** Please do not spend a lot of time trying to decide on what hex code to use. So for instance, for red you can use FF0000 you don't need to find the right shade so trying to do FF1209 is a waste of time. We recommend that you use max (FF), min (00) and average (80) values for each color.

    So for instance, Red would be FF0000, Grey would be 808080, and Yellow would be FFFF00.

    In addition, **we do not want you to spend time calculating the average color for a block,** here is a scheme we want you to use when deciding the colors for blocks.

    Figure out the two colors being mixed together and select one of the following (if applicable) and decide on the hex color to represent the resulting color.

    Red + Blue = Purple

    Black + White=Grey

    Yellow + Blue=Green

    Purple + White= Pink

    Any color + White = increase the value of the non-present color to around 80 (i.e., 50%)

    So for instance:

    Red + White= lighter shade of red so increase the blue and green to 80 = FF8080

    Any color + Black = decrease the value of the non-present colors to around 0(i.e. 0%)

So for instance

Blue + Black = darker shade of blue, so decrease red and green to 00 = 0000FF

After averaging all the 2x2 blocks, the image now looks like this:

000000	FF00FF	8080FF
FFFFFF	FF00FF	FFFFFF
FF00FF	808080	FF8080

The representation using ENCODE would look like this:

03 03
000000 01 FF00FF 01 8080FF 01
FFFFFF 01 FF00FF 01 FFFFFF 01
FF00FF 01 808080 01 FF8080 01

5. How many bytes would be needed to represent the image if we were to use ENCODE?

$$2\ bytes\ +\ \frac{4\ bytes}{run} \times 9\ =\ 38\ bytes$$

6. What would be the % savings?

<u>ENCODE Against Uncompressed Images</u>
$[(110\ bytes\ -\ 38\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 65.5\%$

You could also round this to 67%.

<u>ENCODE Against RLE</u>
$[(106\ bytes\ -\ 38\ bytes)\ \div\ 106\ bytes]\ \times\ 100\%\ =\ 64.2\%$

You could also round this to 64%.

7. Describe a situation in which using ENCODE would be better than using RLE?

If our computer was low on memory, then using ENCODE would be better than RLE.

A situation in which we wanted to know what is the most prevalent color in an image - tinting

Having a square image is not correct :(

8. Describe a situation in which using ENCODE would be worse than using RLE?

If an image had lots of fine detail, then using ENCODE would cause us to lose that.

LOts of different/dissimilar colors, side by side,

9. Is ENCODE an example of a lossy or lossless compression format? Why or why not?

ENCODE is a lossy compression format because it loses information about the original image. Given the ENCODE representation of the image, we can't recreate the original.

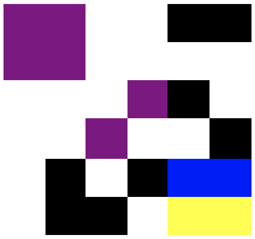
10. What is the largest number of pixels that we can represent with this specification?

Since width and height are both represented as two hexadecimal digits, that means the largest decimal value for width and height would be 255 (0xFF), If both width and height are 255 pixels large, then the largest number of pixels we can represent is 65025.

11. If we wanted to handle a larger number of pixels, what would we need to change in the specification?

We would need more bytes allocated to the width and height.

Version 2



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

06 06

FF00FF 02 FFFFFF 02 000000 02

FF00FF 02 FFFFFF 07

FF00FF 01 000000 01 FFFFFF 03

FF00FF 01 FFFFFF 02 000000 01

FFFFFF 01 000000 01 FFFFFF 01 000000 01 0000FF 02

FFFFFF 01 000000 02 FFFFFF 01 FFFF00 02

$$2\ bytes\ +\ \frac{4\ bytes}{run} \times 20\ runs\ =\ 82\ bytes$$

2. What are the % savings for this image when we use RLE? CASCADING ERROR ONLY

   Uncompressed bytes:

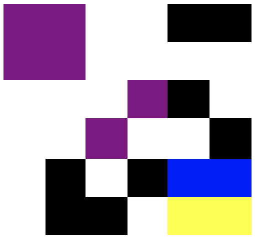
   $$2\ bytes\ +\ \frac{3\ bytes}{colour} \times 36\ pixels\ =\ 110\ bytes$$

   $$[(110\ bytes\ -\ 82\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =25.45\%$$

   You could also round this to 25%

3. What would be the compressed representation of the image using ENCODE and the specification given above?

   After averaging all the 2x2 blocks, the image now looks like this:



FF00FF	FFFFFF	808080
FFFFFF	FF80FF	808080
808080	808080	00FF00

The representation using ENCODE would look like this:

03 03

FF00FF01 FFFFFF 01 808080 01

FFFFFF 01 FF80FF 01 808080 03 00FF00 01

4. How many bytes would be needed to represent the image if we were to use ENCODE?
   CASCADING ERROR

   $2\ bytes\ +\ \frac{4\ bytes}{run} \times 7\ runs\ =\ 30\ bytes$

5. What would be the % savings?

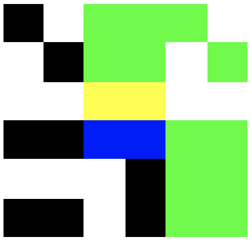
   ENCODE Against Uncompressed Images
   $[(110\ bytes\ -\ 30\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 72.72\%$

   You could also round this to 73%.

   ENCODE Against RLE
   $[(82\ bytes\ -\ 30\ bytes)\ \div\ 82\ bytes]\ \times\ 100\%\ =\ 63.41\%$

   You could also round this to 63%.

Version 3



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

   06 06
   000000 01 FFFFFF 01 00FF00 03 FFFFFF 02

   000000 01 00FF00 02 FFFFFF 01 00FF00 01

   FFFFFF 02 FFFF00 02 FFFFFF 02

000000 02 0000FF 02 00FF00 02

FFFFFF 03 000000 01 00FF00 02

000000 02 FFFFFF 01 000000 01 00FF00 02

$$2\ bytes + \frac{4\ bytes}{run} \times 21\ runs = 86\ bytes$$

2. What are the % savings for this image when we use RLE?

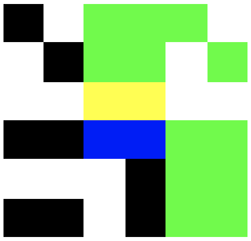
Uncompressed bytes:

$$2\ bytes + \frac{3\ bytes}{colour} \times 36\ pixels = 110\ bytes$$

$$[(110\ bytes - 86\ bytes) \div 110\ bytes] \times 100\% = 21.82\%$$

You could also round this to 22%

3. What would be the compressed representation of the image using ENCODE and the specification given above?



After averaging all the 2x2 blocks, the image now looks like this:

808080	00FF00	80FF80
808080	00FF00	80FF80
808080	808080	00FF00

The representation using ENCODE would look like this:

03 03

808080 01 00FF00 01 80FF80 01

808080 01 00FF00 01 80FF80 01

808080 02 00FF00 01

4. How many bytes would be needed to represent the image if we were to use ENCODE?

$2\ bytes\ +\ \frac{4\ bytes}{run} \times 8\ runs\ =\ 34\ bytes$

5. What would be the % savings?

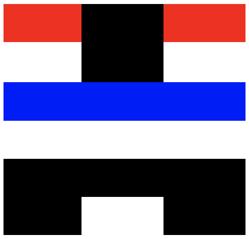
ENCODE Against Uncompressed Images
$[(110\ bytes\ -\ 34\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 69.1\%$

You could also round this to 69%.

ENCODE Against RLE
$[(86\ bytes\ -\ 34\ bytes)\ \div\ 86\ bytes]\ \times\ 100\%\ =\ 60.47\%$

You could also round this to 60%.

Version 4



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

   06 06
   FF0000 02 000000 02 FF0000 02

   FFFFFF 02 000000 02 FFFFFF 02

   0000FF 06

   FFFFFF 06

   000000 08

   FFFFFF 02 000000 02

   $2\ bytes\ +\ \frac{4\ bytes}{run} \times 11\ runs\ =\ 46\ bytes$

2. What are the % savings for this image when we use RLE?

   $[(110\ bytes\ -\ 46\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 58.18\%$
   You could also round this to 58%

3. What would be the compressed representation of the image using ENCODE and the specification given above?

   After averaging all the 2x2 blocks, the image now looks like this:

FF8080	000000	FF8080
8080FF	8080FF	8080FF
000000	808080	000000

The representation using ENCODE would look like this:

03 03

FF8080 01 000000 01 FF8080 01

8080FF 03

000000 01 808080 01 000000 01

4. How many bytes would be needed to represent the image if we were to use ENCODE?

$$2\ bytes\ +\ \frac{4\ bytes}{run} \times 7\ runs\ =\ 30\ bytes$$

5. What would be the % savings?

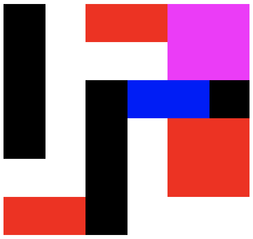
ENCODE Against Uncompressed Images
$[(110\ bytes\ -\ 30\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 72.7\%$

You could also round this to 73%.

ENCODE Against RLE
$[(46\ bytes\ -\ 30\ bytes)\ \div\ 46\ bytes]\ \times\ 100\%\ =\ 34.8\%$

You could also round this to 35%.

Version 5



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

06 06
000000 01 FFFFFF 01 FF0000 02 FF80FF 02

000000 01 FFFFFF 03 FF80FF 02

000000 01 FFFFFF 01 000000 01 0000FF 02 000000 02

FFFFFF 01 000000 01 FFFFFF 01 FF0000 02

FFFFFF 02 000000 01 FFFFFF 01 FF0000 04

000000 01 FFFFFF 03

$$2\ bytes\ +\ \frac{4\ bytes}{run} \times 22\ runs\ =\ 90\ bytes$$

2. What are the % savings for this image when we use RLE?

   Number of bytes to represent image $2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 36\ pixels\ =\ 110\ bytes$

   $[(110\ bytes\ -\ 90\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 18.18\%$

   You could also round this to 18%

3. What would be the compressed representation of the image using ENCODE and the specification given above?

   After averaging all the 2x2 blocks, the image now looks like this:

808080	FF8080	FF80FF
808080	0000FF	FF00FF
FF8080	808080	FF8080

   The representation using ENCODE would look like this:

   03 03

   808080 01 FF8080 01 FF80FF 01

   808080 01 0000FF 01 FF00FF 01

   FF8080 01 808080 01 FF8080 01

4. How many bytes would be needed to represent the image if we were to use ENCODE?

$2\ bytes\ +\ \frac{4\ bytes}{run} \times 9\ runs\ =\ 38\ bytes$

5. What would be the % savings?

ENCODE Against Uncompressed Images

$[(110\ bytes\ -\ 38\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 65.45\%$

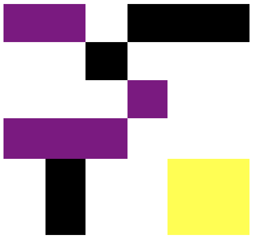
You could also round this to 65%.

ENCODE Against RLE

$[(90\ bytes\ -\ 38\ bytes)\ \div\ 90\ bytes]\ \times\ 100\%\ =\ 57.8\%$

You could also round this to 58%.

Version 6



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

06 06

FF00FF 02 FFFFFF 01 000000 03

FFFFFF 02 000000 01 FFFFFF 06

FF00FF 01 FFFFFF 02

FF00FF 03 FFFFFF 04

000000 01 FFFFFF 02 FFFF00 02

FFFFFF 01 000000 01 FFFFFF 02 FFFF00 02

$2\ bytes\ +\ \frac{4\ bytes}{run} \times 17\ runs\ =\ 70\ bytes$

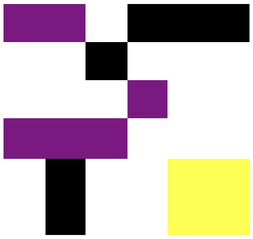
2. What are the % savings for this image when we use RLE?

   $[(110\ bytes\ -\ 70\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 36.4\%$

   You could also round this to 36%

3. What would be the compressed representation of the image using ENCODE and the specification given above?

   After averaging all the 2x2 blocks, the image now looks like this:



FF80FF	808080	808080
FF80FF	FF80FF	FFFFFF
808080	FFFFFF	FFFF00

   The representation using ENCODE would look like this:

   03 03

   FF80FF 01 808080 02

   FF80FF 02 FFFFFF 01

   808080 01 FFFFFF 01 FFFF00 01

4. How many bytes would be needed to represent the image if we were to use ENCODE?

   $2\ bytes\ +\ \frac{4\ bytes}{run} \times\ 7\ runs\ =\ 30\ bytes$

5. What would be the % savings?

   ENCODE Against Uncompressed Images

   $[(110\ bytes\ -\ 30\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 65.5\%$

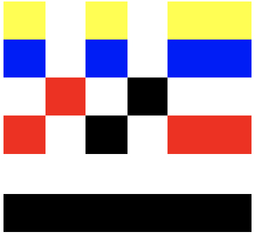
   You could also round this to 67%.

ENCODE Against RLE

$[(70\ bytes\ -\ 30\ bytes)\ \div\ 70\ bytes]\ \times\ 100\%\ =\ 57.1\%$

You could also round this to 57%.

Version 7



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

   06 06

   FFFF00 01 FFFFFF 01 FFFF00 01 FFFFFF 01 FFFF00 02

   0000FF 01 FFFFFF 01 0000FF 01 FFFFFF 01 0000FF 02

   FFFFFF 01 FF0000 01 FFFFFF 01 000000 01 FFFFFF 02

   FF0000 01 FFFFFF 01 000000 01 FFFFFF 01 FF0000 02

   FFFFFF 06

   000000 06

   $2\ bytes\ +\ \frac{4\ bytes}{run}\times\ 22\ runs\ =\ 90\ bytes$

   .

2. What are the % savings for this image when we use RLE?
   (110 bytes - 90 bytes) / 110 bytes X 100% = 18.2%
   You could also round this to 18%.

3. What would be the compressed representation of the image using ENCODE and the specification given above?

   After averaging all the 2x2 blocks, the image now looks like this:

80FF80	80FF80	00FF00
FF8080	808080	FF8080
808080	808080	808080

The representation using ENCODE would look like this:

03 03

80FF80 02 00FF00 01

FF8080 01 808080 01 FF8080 01

808080 03

4. How many bytes would be needed to represent the image if we were to use ENCODE?

$$2\ bytes\ +\ \frac{4\ bytes}{run} \times 6\ runs\ =\ 26\ bytes$$

5. What would be the % savings?

ENCODE Against Uncompressed Images

$$[(110\ bytes\ -\ 26\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 76.4\%$$

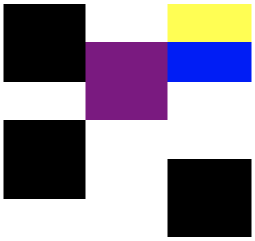
You could also round this to 76%.

ENCODE Against RLE

$$[(90\ bytes\ -\ 26\ bytes)\ \div\ 90\ bytes]\ \times\ 100\%\ =\ 71.1\%$$

You could also round this to 71%.

Version 8



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

   06 06
   000000 02 FFFFFF 02 FFFF00 02

   000000 02 FF00FF 02 0000FF 02

   FFFFFF 02 FF00FF 02 FFFFFF 02

   000000 02 FFFFFF 04

   000000 02 FFFFFF 02 000000 02

   FFFFFF 04 000000 02

   $2\ bytes\ +\ \frac{4\ bytes}{run} \times 16\, runs\ =\ 66\, bytes$

2. What are the % savings for this image when we use RLE?
   (110 bytes - 66 bytes) / 110 bytes X 100% = 40%

3. What would be the compressed representation of the image using ENCODE and the specification given above?

   After averaging all the 2x2 blocks, the image now looks like this:

000000	FF80FF	00FF00
808080	FF80FF	FFFFFF
808080	FFFFFF	000000

The representation using ENCODE would look like this:

03 03

000000 01 FF80FF 01 00FF00 01

808080 01 FF80FF 01 FFFFFF 01

808080 01 FFFFFF 01 000000 01

4. How many bytes would be needed to represent the image if we were to use ENCODE?

$2\ bytes\ +\ \frac{4\ bytes}{run} \times 9\ runs\ =\ 38\ bytes$

5. What would be the % savings?

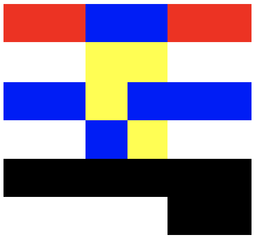
ENCODE Against Uncompressed Images
$[(110\ bytes\ -\ 38\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 65.45\%$

You could also round this to 65%.

ENCODE Against RLE
$[(66\ bytes\ -\ 38\ bytes)\ \div\ 66\ bytes]\ \times\ 100\%\ =\ 42.4\%$

You could also round this to 42%.

Version 9



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

06 06
FF0000 02 0000FF 02 FF0000 02

FFFFFF 02 FFFF00 02 FFFFFF 02

0000FF 02 FFFF00 01 0000FF 03

FFFFFF 02 0000FF 01 FFFF00 01 FFFFFF 02

000000 06

FFFFFF 04 000000 02

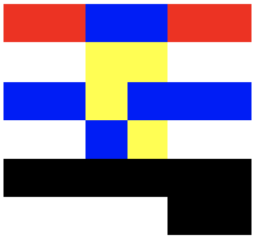
$$2\ bytes\ +\ \frac{4\ bytes}{run} \times 16\ runs\ =\ 66\ bytes$$

2. What are the % savings for this image when we use RLE?

$$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 36\ pixels\ =\ 110\ bytes$$

(110 - 66) / 110 * 100 = 40% savings

3. What would be the compressed representation of the image using ENCODE and the specification given above?



After averaging all the 2x2 blocks, the image now looks like this:

FF8080	00FF00	FF8080
8080FF	00FF00	8080FF
808080	808080	000000

The representation using ENCODE would look like this:

03 03

FF8080 01 00FF00 01 FF8080 01

8080FF 01 00FF00 01 8080FF 01

808080 02 000000 01

4. How many bytes would be needed to represent the image if we were to use ENCODE?

$2\ bytes\ +\ \frac{4\ bytes}{run} \times 8\ runs\ =\ 34\ bytes$

5. What would be the % savings?

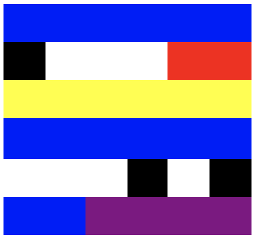
   ENCODE Against Uncompressed Images
   $[(110\ bytes\ -\ 34\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 69.1\%$

   You could also round this to 69%.

   ENCODE Against RLE
   $[(66\ bytes\ -\ 34\ bytes)\ \div\ 66\ bytes]\ \times\ 100\%\ =\ 48.48\%$

   You could also round this to 48%.

Version 10



1. If we were to use RLE with this new specification, how many bytes would it take to represent the image?

   06 06

   0000FF 06

   000000 01 FFFFFF 03 FF0000 02

   FFFF00 06

   0000FF 06

   FFFFFF 03 000000 01 FFFFFF 01 000000 01

   0000FF 02 FF00FF 04
   $2\ bytes\ +\ \frac{4\ bytes}{run} \times 12\ runs\ =\ 50\ bytes$

2. What are the % savings for this image when we use RLE?

$2\ bytes\ +\ \frac{3\ bytes}{pixel} \times 36\ pixels\ =\ 110\ bytes$

(110 - 50) / 110 * 100 = 55% savings

3. What would be the compressed representation of the image using ENCODE and the specification given above?

After averaging all the 2x2 blocks, the image now looks like this:

0000FF	8080FF	FF00FF
00FF00	00FF00	00FF00
8080FF	FF00FF	FF00FF

The representation using ENCODE would look like this:

03 03

0000FF 01 8080FF 01 FF00FF 01

00FF00 03

8080FF 01 FF00FF 02

4. How many bytes would be needed to represent the image if we were to use ENCODE?

$2\ bytes\ +\ \frac{4\ bytes}{run} \times 6\ runs\ =\ 26\ bytes$

5. What would be the % savings?

ENCODE Against Uncompressed Images

$[(110\ bytes\ -\ 26\ bytes)\ \div\ 110\ bytes]\ \times\ 100\%\ =\ 76.36\%$

You could also round this to 76%.

ENCODE Against RLE

$[(50\ bytes\ -\ 26\ bytes)\ \div\ 50\ bytes]\ \times\ 100\%\ =\ 48\%$

You could also round this to 48%.

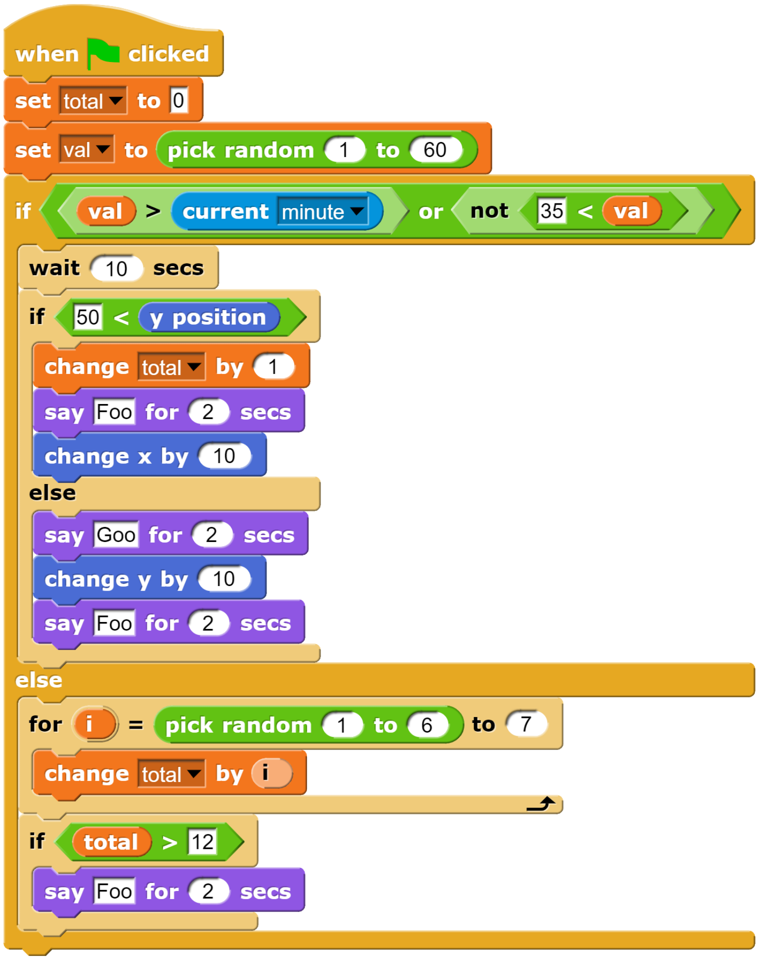
Reading Snap

For the code block depicted in the image shown, when does the sprite ONLY say “Foo”?

In other words, write out **ALL** the conditions that must be met for the Sprite to **ONLY** say “Foo”

Morning

Version 1



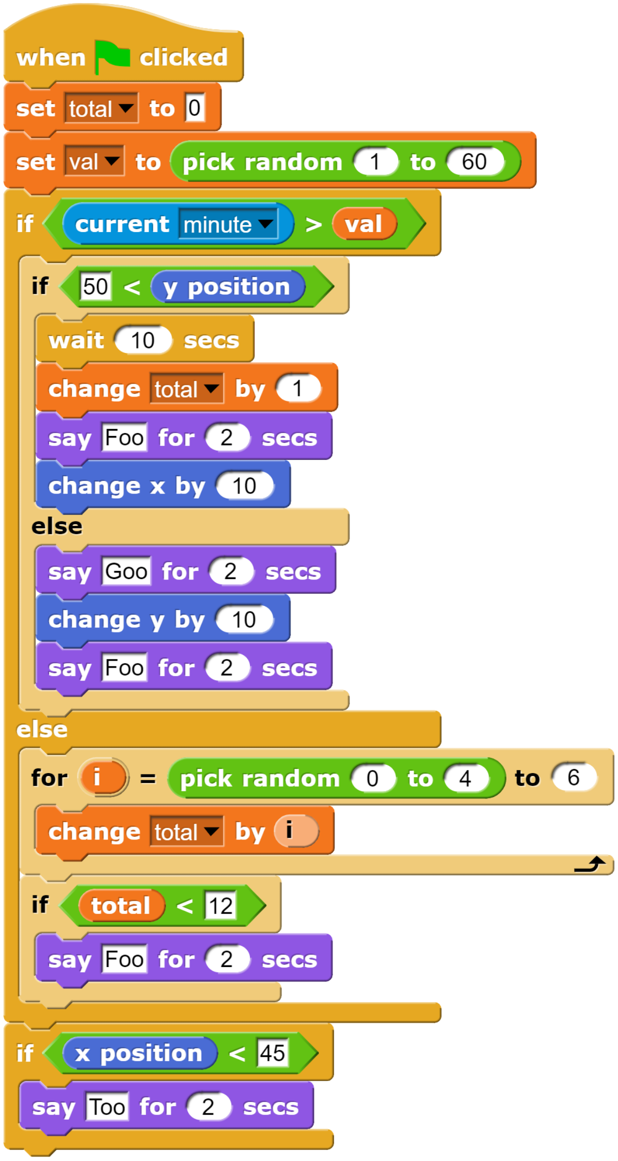
```
when ⚑ clicked
set total to 0
set val to pick random 1 to 60
if val > current minute or not 35 < val
    wait 10 secs
    if 50 < y position
        change total by 1
        say Foo for 2 secs
        change x by 10
    else
        say Goo for 2 secs
        change y by 10
        say Foo for 2 secs
else
    for i = pick random 1 to 6 to 7
        change total by i
    if total > 12
        say Foo for 2 secs
```

- random ”val” must be greater than current minute (1-60) or less than 35
- y value must be greater than 50

OR

- if random “val” is less than current minute, “i” must be any number from 1-6
- if student refers to “current minute” as a specific time, **subtract 1**

Version 2



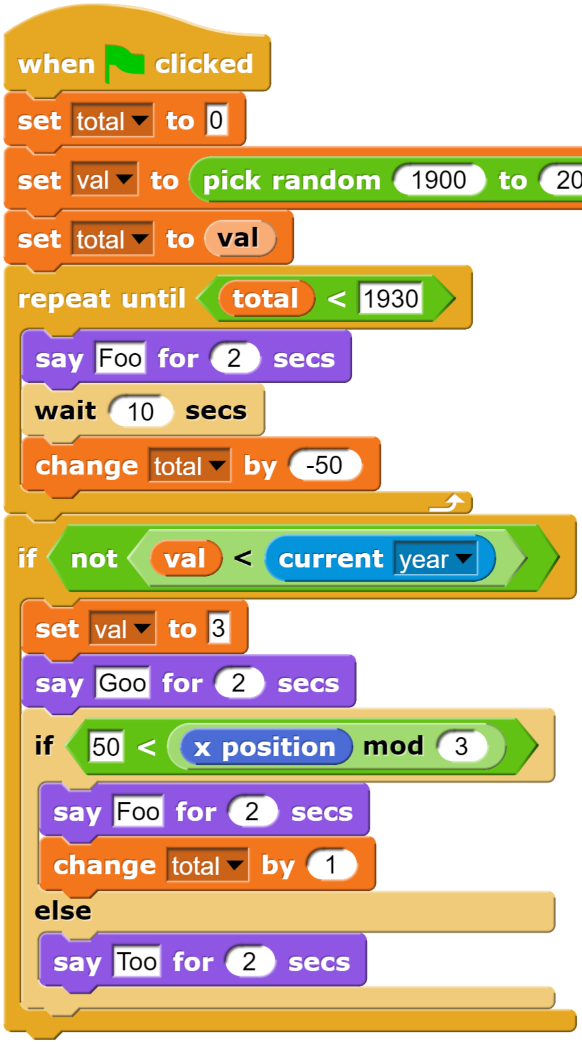
```
when ⚑ clicked
set total to 0
set val to pick random 1 to 60
if current minute > val
    if 50 < y position
        wait 10 secs
        change total by 1
        say Foo for 2 secs
        change x by 10
    else
        say Goo for 2 secs
        change y by 10
        say Foo for 2 secs
else
    for i = pick random 0 to 4 to 6
        change total by i
    if total < 12
        say Foo for 2 secs
if x position < 45
    say Too for 2 secs
```

- current minute needs to be greater than the chosen “val”
- y position must be greater than 50
- x position must be greater than or equal to 45

- if student refers to “current minute” as a specific time, **subtract 1**

Afternoon

Version 1



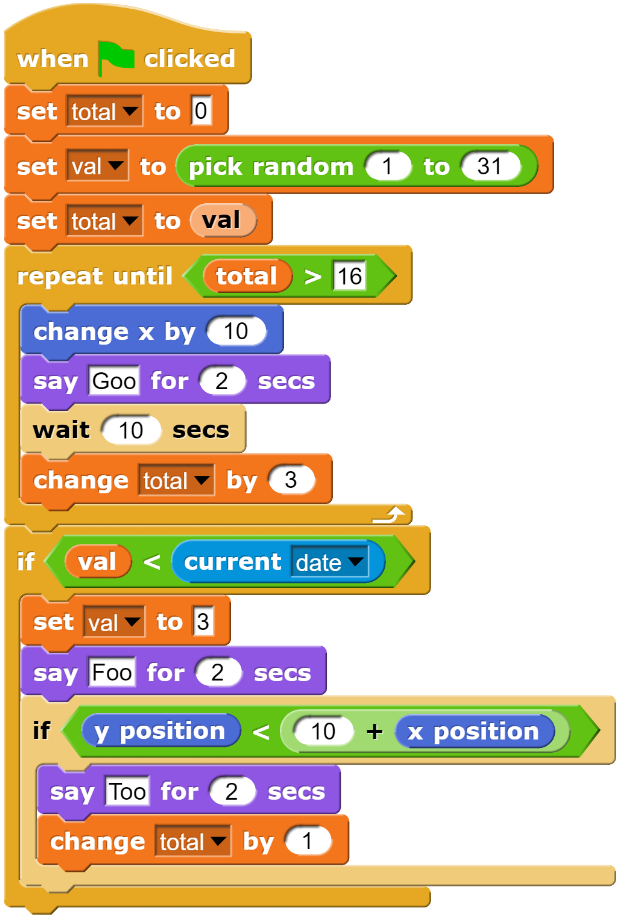
```
when ⚑ clicked
set total to 0
set val to pick random 1900 to 2020
set total to val
repeat until total < 1930
    say Foo for 2 secs
    wait 10 secs
    change total by -50
if not val < current year
    set val to 3
    say Goo for 2 secs
    if 50 < x position mod 3
        say Foo for 2 secs
        change total by 1
    else
        say Too for 2 secs
```

- Val has to be a number between 1930 and 2020. Otherwise, the repeat-until loop will not run so the sprite cannot say foo.

- We cannot ever let the if-statement after the repeat-until loop run because once it does, then the sprite will say “Goo”. Since we cannot let the outer if-statement execute, the inner if-statement can never be executed either. We do not have to consider the x position for this reason.

- -1 mark if the answer assumes a particular value for the variable year.

Version 2



```
when 🏳 clicked
set total to 0
set val to pick random 1 to 31
set total to val
repeat until total > 16
    change x by 10
    say Goo for 2 secs
    wait 10 secs
    change total by 3
if val < current date
    set val to 3
    say Foo for 2 secs
    if y position < 10 + x position
        say Too for 2 secs
        change total by 1
```

- Randomly picked number must be above 16 but ***less than the current date***
  - 17-19
  - 17-31
- y > 10 + x