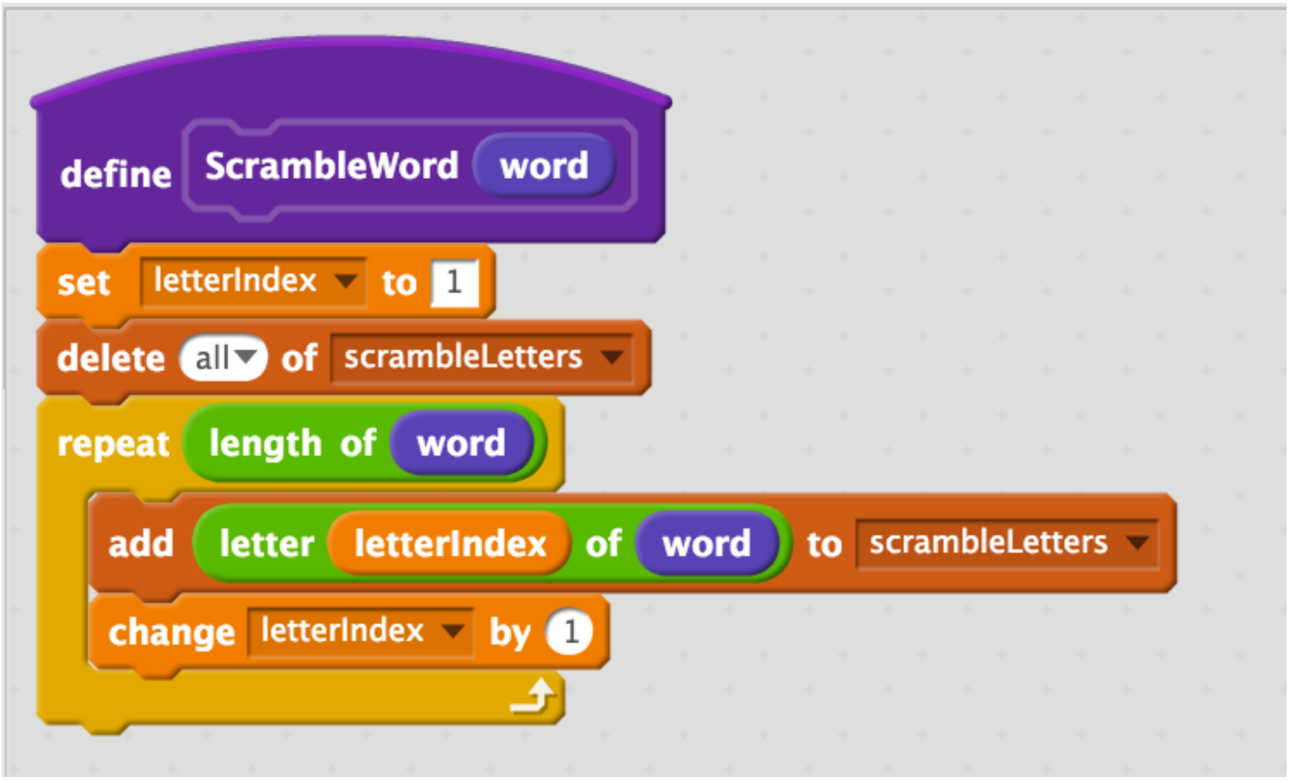
Anagram Advent Calendar - Scratch

Step 1 - Start Building the Word Scrambler Function

This step adds code to the sprite which will act as our presents in the calendar. Add all the following code to the sprite that is created with the new project.
Once you have done this, the function should look like this.



```
define ScrambleWord word
set letterIndex to 1
delete all of scrambleLetters
repeat length of word
    add letter letterIndex of word to scrambleLetters
    change letterIndex by 1
```

* Under More Blocks, click Make a Block
* Type the name ScrambleWord and add a single String input called word.
* Under Data, click Make a List, call it scrambleLetters. This function will put each letter into this list separately. A loop can then randomly pick them back out to assemble the scrambled word.
* Create another variable to contain the return value for the Scramble function. Give it the name returnScrambleWord.
* Create another variable to contain the index of the letter as we loop through. Give it the name letterIndex.
* From Data, drag set <letterIndex> to <0> block into our function. Type 1 into the white box to initialise the letter index to point to the first character in our word. Letters in Scratch strings are indexed from 1 rather than 0.
* Before collecting the letters, it will be important to clear out the list of letters. So from Data, drag a delete <1> of scrambleLetters into our function. Use the drop down to select <all>. This will clear out the list.
* From Control, drag a repeat <10> block into our function.
* From Operators, drag a length of <world> block into the range limit of our repeat block.
* Drag the word variable from the function into the length of <world> block.
* From Data, drag an add <thing> to <scrambleLetters> block into our loop. This will be programmed to pick letters from the word.
* From Operators drag letter <1> of <world> block into the add <thing> box.
* From Data, drag the letter index into the <1> of the new letter selection operator.
* Drag the word variable from the function into the <world> of the new letter selection operator.
* After the letter has been added, we should increment the letterIndex so the loop can move onto the next letter. From Data, drag the change <letterIndex> by <1>

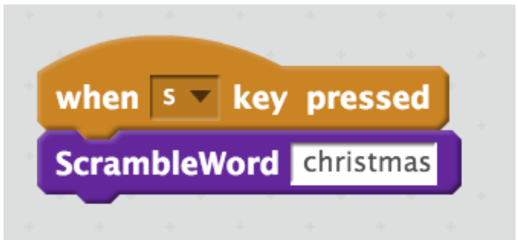
Step 2 - Test the Scramble Function

Before we go any further, we should check to see how the function is working. Currently it will just pick all the letters out of a word and put them in a list. To test this function, we will now write a test block that can be called by pressing a key.

* From Events, drag a when <space> key is pressed block into the code area. Change the key to something other than space, perhaps <s> for scramble.
* To call our function, from More Blocks, drag the ScrambleWord block under our new button press event.
* Type in a word to the white box of the ScrambleWord block. Then try pressing <s> and check that the list of scrambleLetters is being filled in properly.

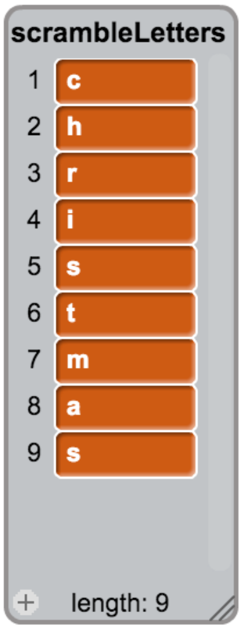
As we are writing the final the scramble function, you can keep using this function to test that it is working as you go along. This sort of testing is very useful to understanding how the development is progressing.

The function should look like this.



```
when [s] key pressed
ScrambleWord [christmas]
```

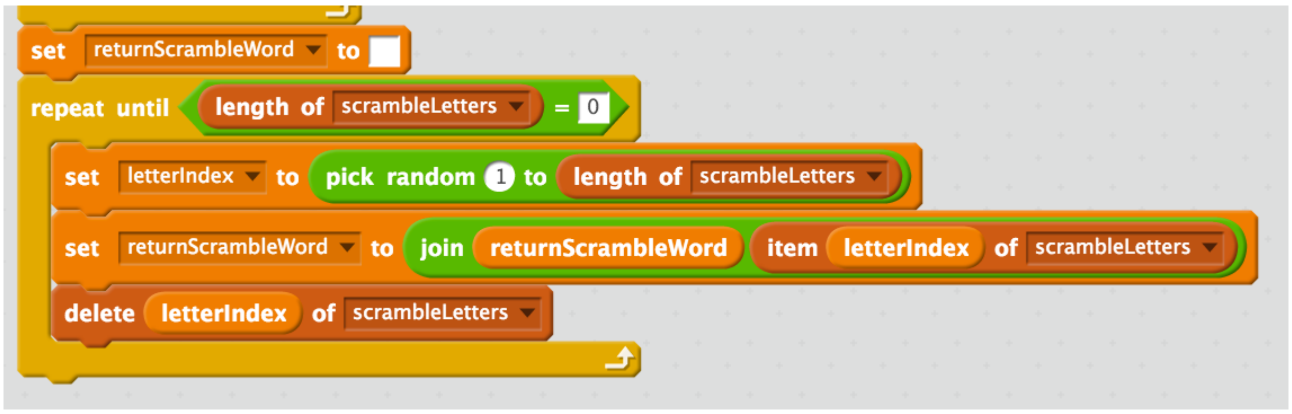
When run, the list should appear in the game view like this.



```
scrambleLetters
1 c
2 h
3 r
4 i
5 s
6 t
7 m
8 a
9 s
+ length: 9
```

Step 3 - Finish the Scramble Word function

The ScrambleWord function currently assembles the letters into a list. It is now time to compose them into a ScrambledWord.
Once completed, the new part of the code should look like this.



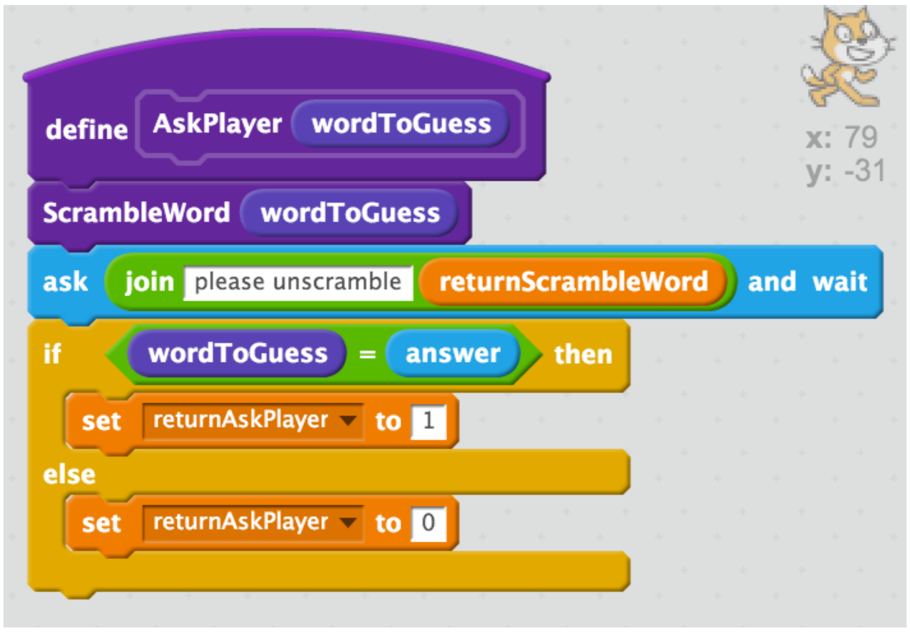
```
set returnScrambleWord to [ ]
repeat until <(length of scrambleLetters) = 0>
    set letterIndex to (pick random 1 to (length of scrambleLetters))
    set returnScrambleWord to (join (returnScrambleWord) (item (letterIndex) of scrambleLetters))
    delete (letterIndex) of scrambleLetters
```

* First we must set the scrambled word to empty string. We created a variable earlier that I called returnScrambleWord. From Data drag set <returnScrambledWord> to <0> block underneath our loop. Delete the <0> so that the variable is set to an empty string. We will be adding letters to it as we go along.
* We will now use a loop to work through the list of letters randomly. For this we shall use a while loop. From Control drag a repeat until <> block into our function.
* The condition will be the number of items in our list is more than zero, so from Operators, drag an equals block into our repeat condition.
* In the left hand box, we must put the number of items in our letters list. From Data drag length of <scrambleLetters> into the right hand box.
* In the right hand white box, type 0. This then sets the loop to continue until we have run out of letters.
* We now need to pick a random index, we shall re-use the letterIndex variable that we created earlier. We will set it to a random value. From Data drag a set <letterIndex> to <0> block into our loop.
* From Operators, drag a pick random from <1> to <10> block into our new set block.
* The range of this random number should be 1 to <number of items in scrambleLetters>. So from Data drag a length of scrambleLetters block into the upper range of the random block.
* The letter index is now set to a valid index of a letter in our list. In order to put that letter onto our scrambled word we must do a join. From Data drag a set <returnScrambleWord> to <0> block into our loop.
* From Operators, drag a join <hello> <world> block into the new set block.
* The first argument of this join block should be the existing value of scramble word. From Data, drag the returnScrambleWord into the left hand white box of this new join block.
* Now we must pick the random letter from our list and append it to the scrambled word. From Data drag item <1> of <scrambleLetters> block into the right hand white box of the join.
* Rather than item <1> though, we want to use the random number we generated earlier. So from Data drag the value of letterIndex into the item <letterIndex> of <scrambleLetters> block.
* Now that a letter has been picked, we should remove it from the list. This will ensure that the list gets smaller on each loop round. Once we have run out of letters then the returned word will be complete and the script can finish. So from Data, drag delete <1> of <scrambleLetters> block into our loop.
* Replace the <1> with <letterIndex> by dragging the value of letterIndex from Data into the new delete block.

The word scramble should be complete. Run your test again to check that the returnScrambleWord gets set correctly. You could add a say <returnScrambleWord> block to your test in order to get the sprite to read out the scrambled value.

Step 4 - Build Player Interaction Function

Now that we can scramble words, we should build the user interaction that lets the player solve the anagram. We will initiate a simple solver on pressing a key. We will build the user interaction into a function called AskPlayer.
Once complete, the function should look like this.



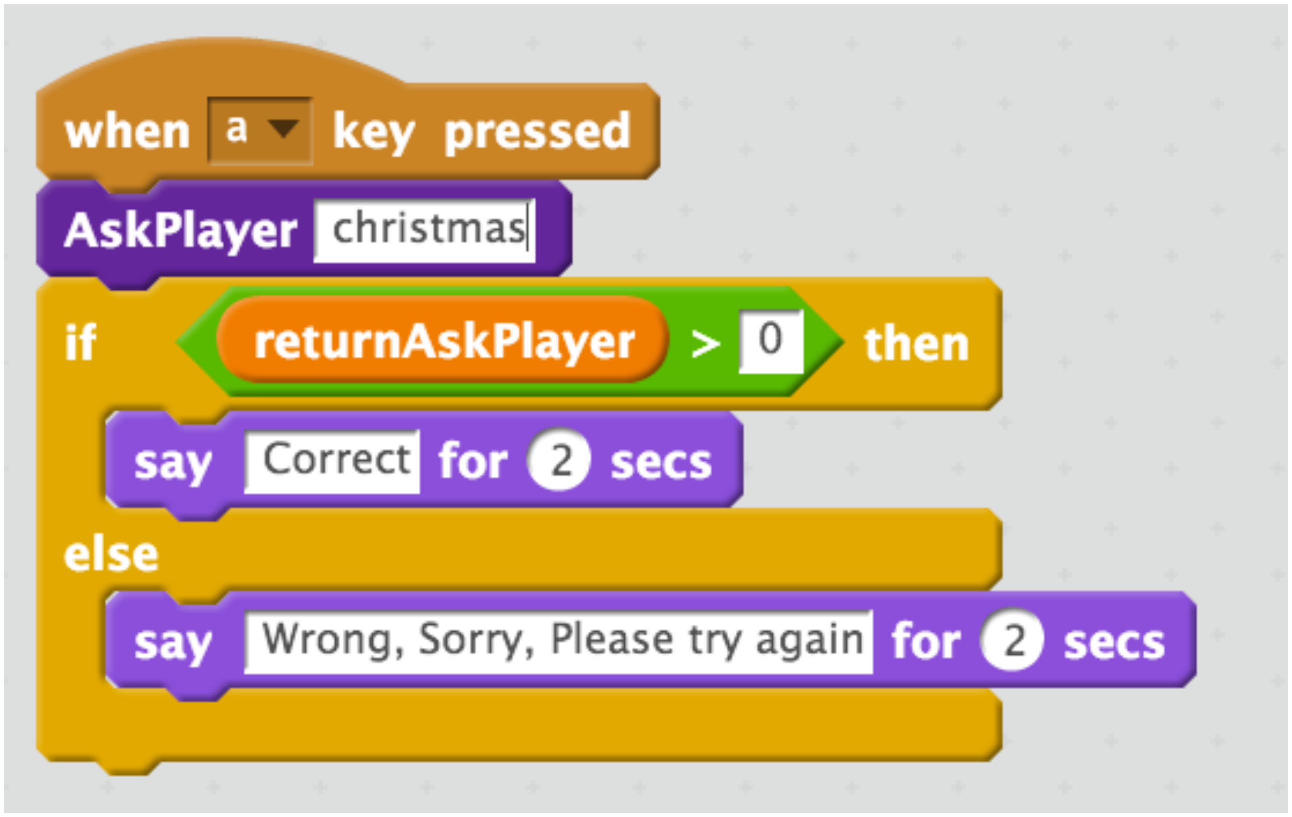
```
define AskPlayer wordToGuess
ScrambleWord wordToGuess
ask join please unscramble returnScrambleWord and wait
if wordToGuess = answer then
    set returnAskPlayer to 1
else
    set returnAskPlayer to 0
```

* From More Blocks, click on Make a Block. Give it the name AskPlayer and give it a single string input called wordToGuess. This will the unscrambled word and the player will be expected to type this in response to the scrambled word.
* From More Blocks, drag our ScrambleWord block into this function.
* Drag the value of wordToGuess from the function header into this block. This will then pass the wordToGuess in for scrambling.
* In order to ask the user to make a guess, from Sensing, drag ask <whats your name> and wait block into our function.
* We want to ask the player to unscramble a word, so we should join the text ‘please unscramble’ to our scrambled word. From Operators drag a join <hello> <world> block into our ask block.
* In the left hand box type the text ‘please unscramble ‘ and then drag the returnScrambleWord variable into the right hand box.
* When the player types in a response and presses enter, the program will be allowed to continue. We can then compare the answer they have given with the wordToGuess to see if they were correct. From Control drag an if-else block into our function.
* To test the equality of the answer with our wordToGuess, from Operators drag an equals block into our if condition.
* Into the left hand box of the equals condition, drag wordToGuess.
* From Sensing, drag answer into the right hand box. The condition is now testing that the answer given matches the word we scrambled.
* In order to indicate the return value of this function, create a new variable. From Data click Make a Variable and call it returnAskPlayer.
* Inside our if-else, drag a set <returnAskPlayer> to <0> into both the if and the else conditions.
* If the word was correct, the return value should be set to 1, if it was incorrect then it should be set to 0.

This completes the player interaction function. As before we will now write a test to check that it is working.

Step 5 - Test Player Interaction

We are going to write a function to test the player interaction, once complete it should look like this.



```
when [a ▾] key pressed
AskPlayer [christmas]
if <(returnAskPlayer) > [0]> then
    say [Correct] for (2) secs
else
    say [Wrong, Sorry, Please try again] for (2) secs
```

* From Events drag a when <space> key is pressed into the code area. Change this to <a> for anagram.
* Form More Blocks, drag in our AskPlayer block into this new function. Type in a word for the player to guess, I have used ‘christmas’ in this example.
* If you now run the test by pressing <a> the game should ask you for your answer. But when the user enters their answer nothing will happen. In order to fix this, from Control drag another if-else block into our new function.
* We will check the value of returnAskPlayer, if it is more than zero, then the player got the answer right. From Operators drag a more than block into our condition.
* The right hand box should contain 0, the left hand box should contain the value of returnAskPlayer (drag this from Data)
* If the player gets it right, let us just have the sprite say <correct> for <2> secs, if they get it wrong, we can have it say <please try again> for <2> secs. You can find the say block under looks.

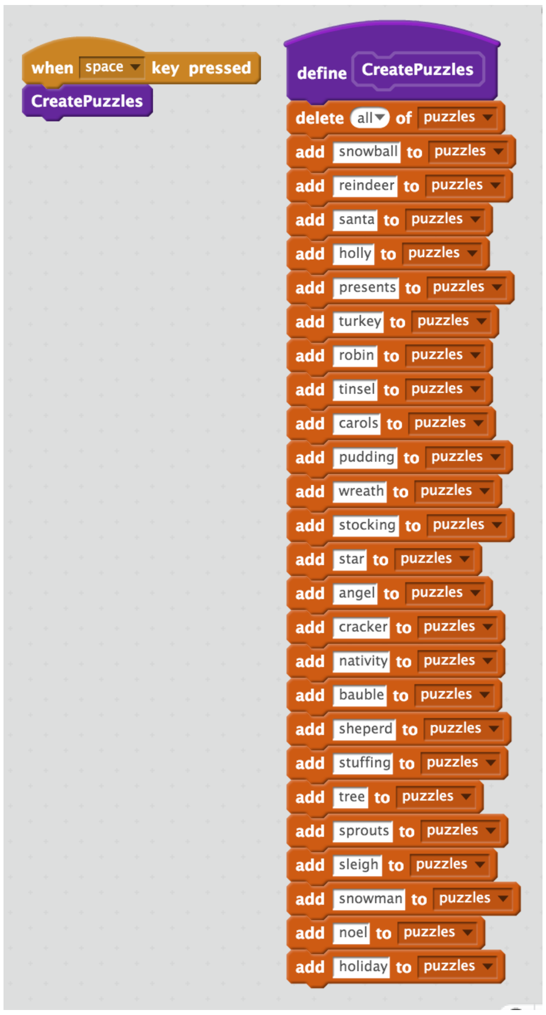
Run the test again and type in the correct unscrambled word. The sprite should congratulate you. Now to test it, get it wrong deliberately and the sprite should politely ask you to try again.

Step 6 - Create List of Puzzles

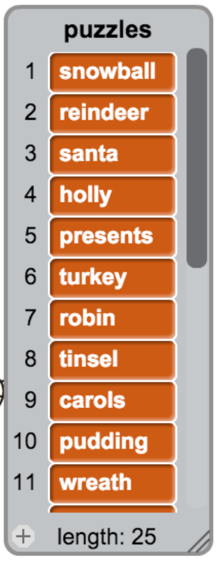
In order to provide a different puzzle for each day leading up to Christmas, we must now create a list of puzzles with 25 items in. We will later be creating 25 clones of our sprites to act as presents.

* Under Data, click on Make a List, call it puzzles.
* Under More Blocks, click on Create a New Block. Give it the name CreatePuzzles, it won't need any parameters.
* First thing to do is to ensure the list of puzzles is clear (this code will be run repeatedly). So from Data drag delete <all> of <puzzles> into our new function. Use the drop down boxes to set <all> and the correct list variable.
* We now need 25 separate puzzles, so drag the add <thing> to <puzzles> block from Data into our function 25 times. To save time, drag it 5 times, then use duplicate 5 times. It is always good to find lazy ways to save clicking.
* Starting at the first <thing> type in 25 separate words, you can use the TAB key to work through them in order to do it quicker. Each word will be a puzzle on a different day.
* In order to kick this off, give yourself a new button handler. Under Events, drag a when <space> key pressed into our code area.
* Inside this function, drag the Create Puzzles piece from More Blocks.
* Now try using the <space> key to populate the list. The list should be visible in the game area.

The code to generate the list, and to call the generate function should look like this.



```
when space key pressed
CreatePuzzles

define CreatePuzzles
delete all of puzzles
add snowball to puzzles
add reindeer to puzzles
add santa to puzzles
add holly to puzzles
add presents to puzzles
add turkey to puzzles
add robin to puzzles
add tinsel to puzzles
add carols to puzzles
add pudding to puzzles
add wreath to puzzles
add stocking to puzzles
add star to puzzles
add angel to puzzles
add cracker to puzzles
add nativity to puzzles
add bauble to puzzles
add sheperd to puzzles
add stuffing to puzzles
add tree to puzzles
add sprouts to puzzles
add sleigh to puzzles
add snowman to puzzles
add noel to puzzles
add holiday to puzzles
```



```
puzzles
1  snowball
2  reindeer
3  santa
4  holly
5  presents
6  turkey
7  robin
8  tinsel
9  carols
10 pudding
11 wreath
+ length: 25
```

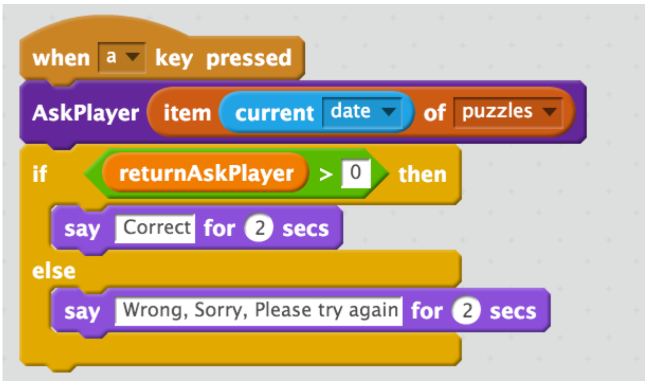
Step 7 - Solve Today's Puzzle

We will now use Scratches ability to determine the current time/date to present the correct puzzle for the day that the player has logged into our game. For this we will re-use the test function we wrote in test 5 (when player presses <a>). Instead of presenting a fixed string, we will pick the correct puzzle from our list.

* From Data, drag the item <1> of <puzzles> block into the existing AskPlayer function call in our function.
* Instead of item <1> though, we want to use the current day. From Sensing, drag current <date> into the item selection block. Use the drop down menu to select <date> it usually has <minute> by default.

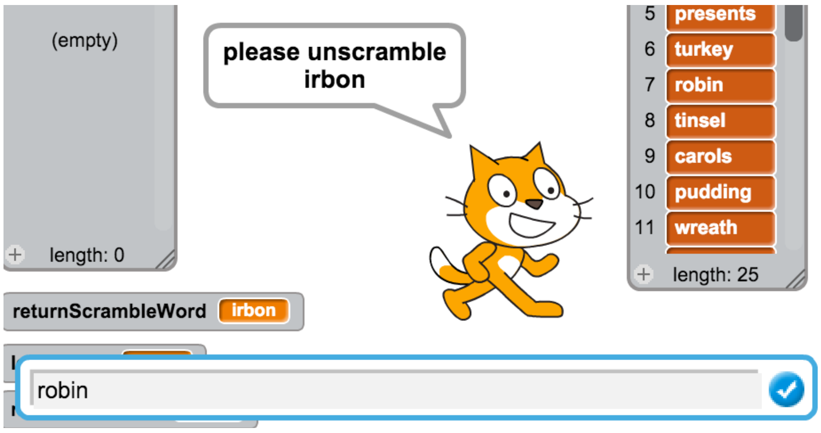
Run this function now by pressing <a>. You should find that it picks the correct date puzzle, scrambles it and asks you to type in the unscrambled word. From that point it will say correct/ incorrect as appropriate.

The edited function should look like this



```
when a key pressed
AskPlayer (item (current date) of puzzles)
if <(returnAskPlayer) > 0> then
    say Correct for 2 secs
else
    say Wrong, Sorry, Please try again for 2 secs
```

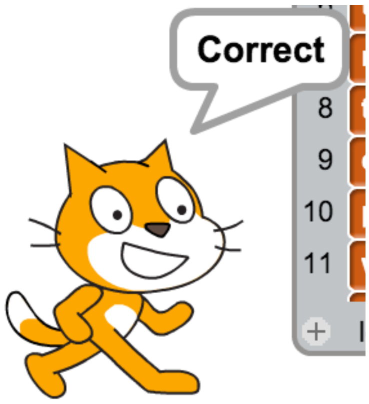
When run the code should go through the scrambling function and present the puzzle, type the correct answer and check what happens.



(empty)
length: 0
please unscramble irbon

5	presents
6	turkey
7	robin
8	tinsel
9	carols
10	pudding
11	wreath

length: 25
returnScrambleWord irbon
robin

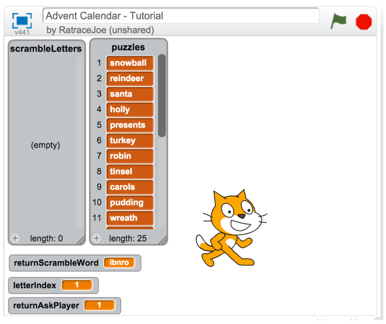


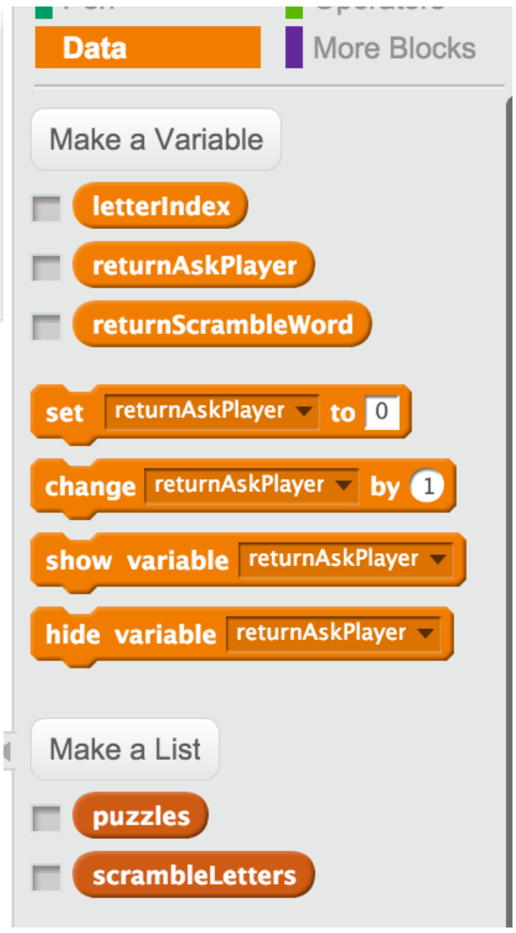
Correct

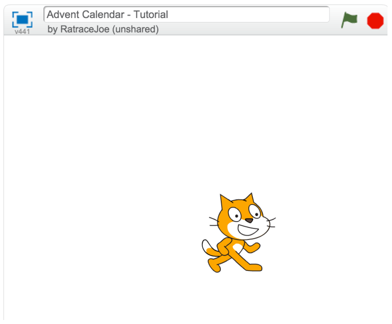
Step 8 - Clean up Game Area

The next stage will create the rows of presents that the player can click on. Before we do this though we need to clean up the game area. By default Scratch shows the variables on the screen. They can be hidden by using the tick boxes under the Data area.

Under Data, uptick all the variables until the game area is just showing the sprite.





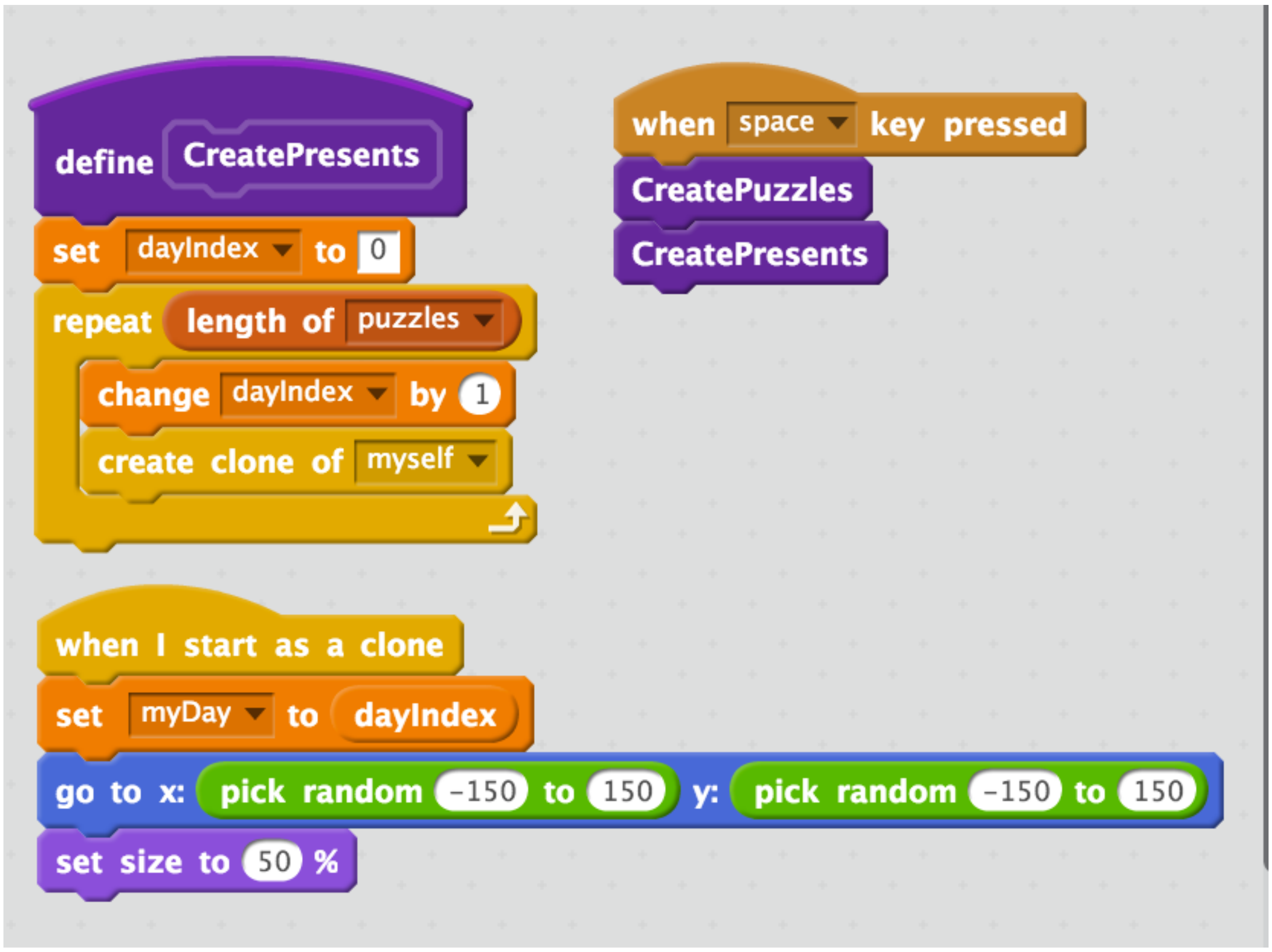


Step 9 - Create Presents

This section will create a new block that will clone our sprite 25 times. The sprite will have two costumes.

1. A present that can be opened upon clicking (like an advent calendar)
2. The cat sprite (or another of your choosing) to act as the opened present

When the following instructions are completed the new code will look like this.

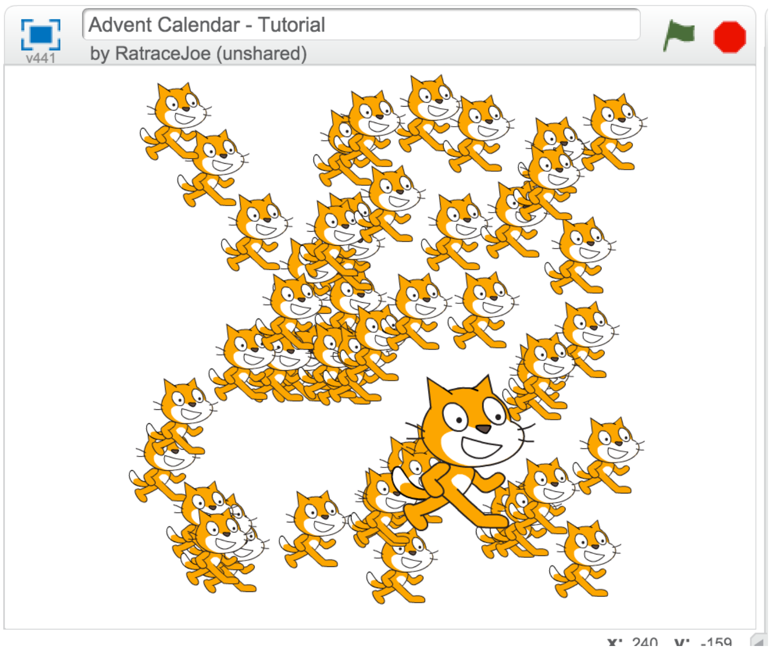


```
define CreatePresents
set dayIndex to 0
repeat length of puzzles
    change dayIndex by 1
    create clone of myself

when I start as a clone
set myDay to dayIndex
go to x: pick random -150 to 150 y: pick random -150 to 150
set size to 50 %

when space key pressed
CreatePuzzles
CreatePresents
```

* Under More Blocks, click Make a Block, give it the name CreatePresents. It doesn't need any parameters.
* Each sprite clone will be given its own ID, so create a new variable, scoped to the sprite, to hold the identity. We will need two variables. One to control the loop and one to store the id. Under Data, click Make a Variable, check the box For This Sprite Only, give it the name myDay. Then create another variable called dayIndex.
* From Data, drag a set <dayIndex> to <0> into our new function.
* We now need a loop to create the clones. From Control drag a repeat <10> into our function.
* The loop should create a present for each puzzle we have defined, so from Data, drag length of <puzzles> into the white box of the repeat block.
* To get the loop to increment the day each time, drag a change <dayIndex> by <1> into our loop. This will ensure the first clone starts with a myDay value of 1 for the 1st of December.
* To create the clones, from Control drag a create clone of <myself> into our loop.
* When a sprite is created as a clone, it should take on the value of dayIndex into its myDay variable. Each clone has its own copy of myDay (if you clicked For This Sprite Only). From Control drag When I Start as a Clone into the code area.
* From Data drag a set <myDay> to <0> block into our new clone initialiser.
* From Data, drag the value of dayIndex into this new setter block.

* If we create the clones now, they all stack up on top of each other, which isn't very helpful, so for each clone we will set a random position. Later we will tidy this up. From Motion drag a go to x:<x> y:<y> block into our clone initialiser.
* From Operators drag a pick random from <1> to <10> into both the x and y values of our position setter. Change the random range from -150 to 150.
* In addition, let us set the size down a bit so they don't overlap so much. From Looks drag set size to <100> % into our clone initialiser, change this block to set to 50%.
* In order to create the presents just after creating the puzzles, we should call CreatePresents after calling CreatePuzzles on pressing the <space> bar. We created a function in step 6 to do this. So from More Blocks, drag a call to CreatePresents into the <space> bar function, just after the call to CreatePuzzles.

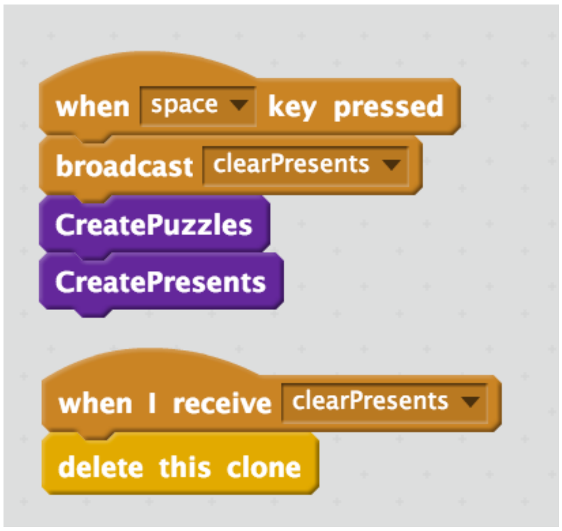
Now run the code and you should see the following.



Step 10 - Control the Creation of Clones

If you press your <space> bar a few times, you will see that it keeps adding clones to the game area. We only ever need 25 clones, so we would like it to delete all clones before creating new ones. In order to do this, we shall create a broadcast to send before new clones are created. Any existing clones can react to that broadcast by deleting themselves.

When you have done this step, the affected sections of code will look like this.



```
when space key pressed
broadcast clearPresents
CreatePuzzles
CreatePresents

when I receive clearPresents
delete this clone
```

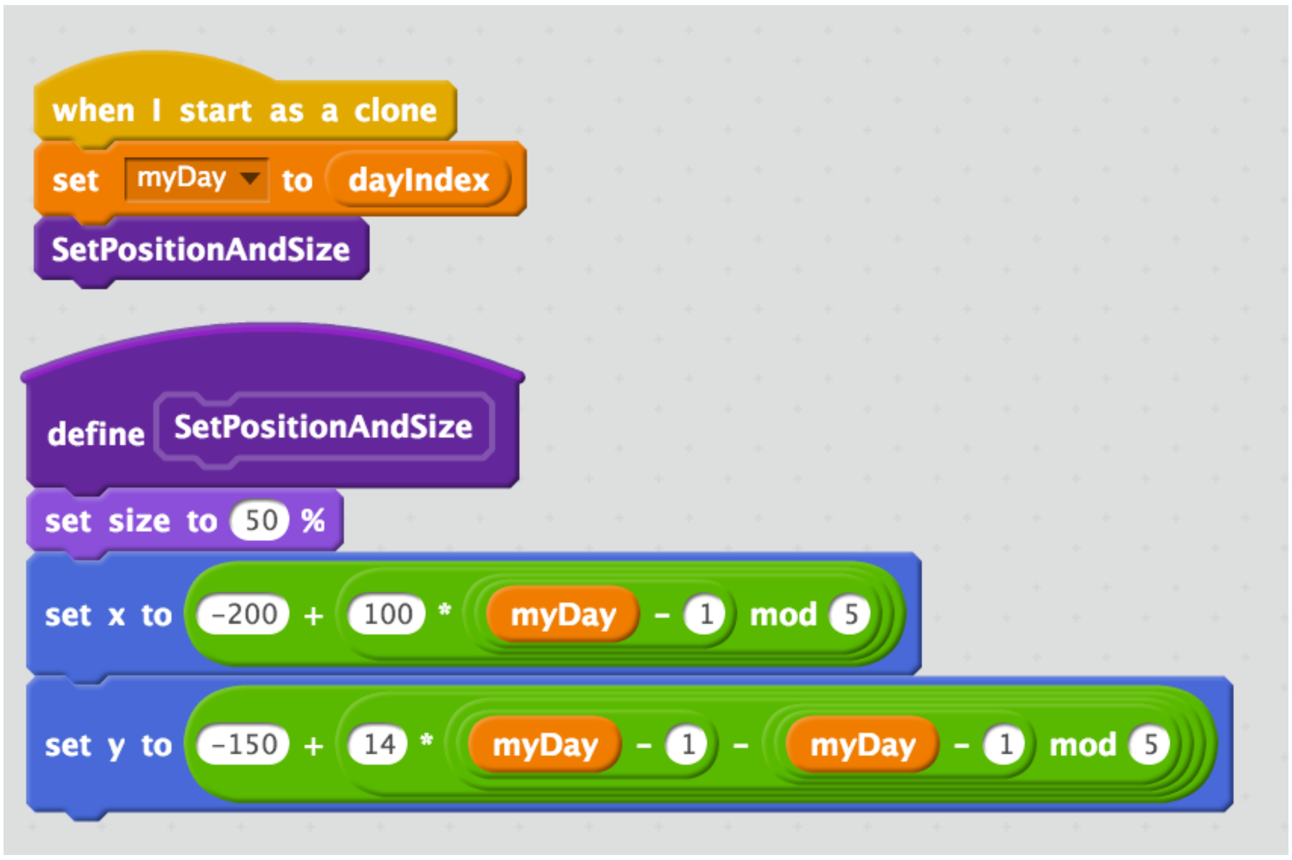
* Under the <space> key press, before the call to CreatePuzzles, drag a broadcast <message1> from Events.
* Right click on your new broadcast <message1> and rename the broadcast to something helpful. I have used clearPresents.
* Also from Events, drag a When I receive <clearPresents> into the code area.
* From Control, drag a delete this clone block into our broadcast receiver.

Now try pressing the space bar a few times, you should notice that any existing clones are deleted before new ones are created.

Step 11 - Position the Clones

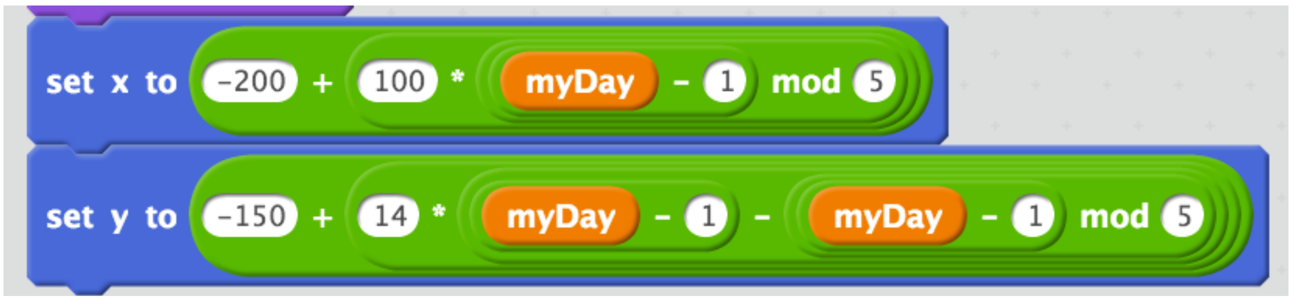
The clones are currently randomly positioned in the game area which is not very helpful. We shall now calculate a position for each clone so that they line up in a 5x5 grid. In order to calculate the positions of each clone, we must do some maths on the myDay variable.

The finished code for this section will look like this



```
when I start as a clone
set myDay to dayIndex
SetPositionAndSize

define SetPositionAndSize
set size to 50 %
set x to (-200 + (100 * (((myDay - 1) mod 5))))
set y to (-150 + (14 * ((myDay - 1) - ((myDay - 1) mod 5))))
```

* Under More Blocks, click on Make a Block, given the new block a name like SetPositionAndSize.
* Under your existing When I Start as a Clone block, you should see a go to x: y: block and a set size block. Replace this with a call to your SetPosition variable.
* Drag the set size to 50% block into our new SetPositionAndSize block. If you lost it, drag a new one from Looks.
* From Motion, we will drag a set x to <0> block and a set y to <0> block into our new function. We will calculate x and y separately because each one will be fairly complex.
* Now we must build the mathematical functions for calculating x and y. We will be subtracting 1 from myDay when it is used because we need to first item in each row to have a zero offset. We are using the mod function to control the row size of 5.

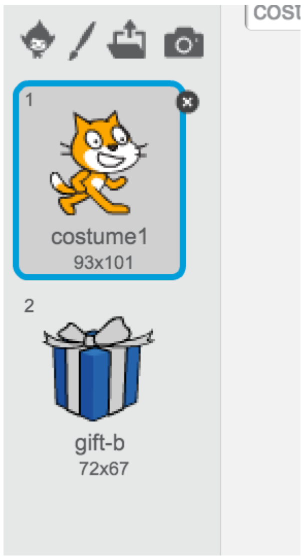


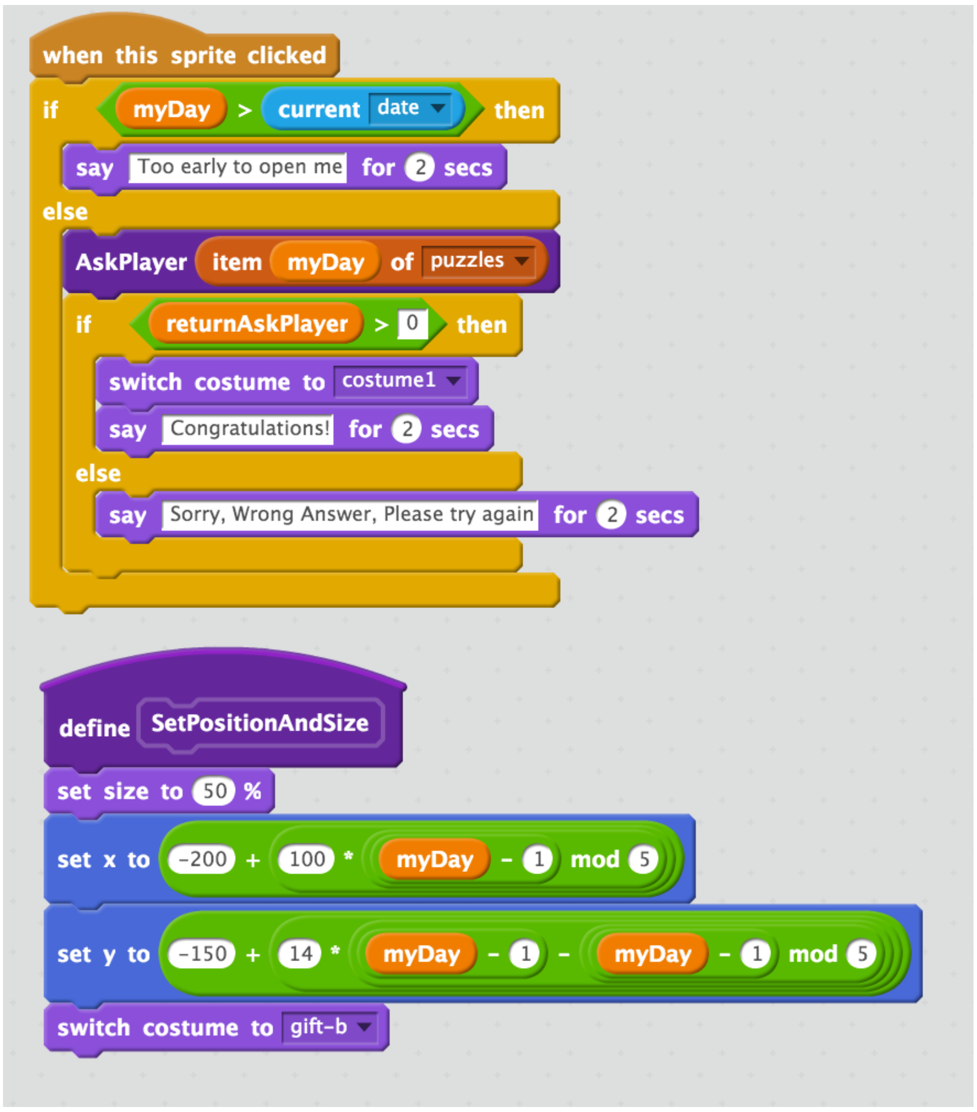
```
set x to (-200 + (100 * (((myDay - 1) mod 5))))
set y to (-150 + (14 * ((myDay - 1) - ((myDay - 1) mod 5))))
```

Step 12 - Opening the Presents

When the clones are created, they should appear as presents. When the user clicks on a present it should present the puzzle of the day. If they user gets the puzzle correct the present should open.

If you correctly follow these instructions, the code should end up looking like this. (Also showing the 2 costumes).





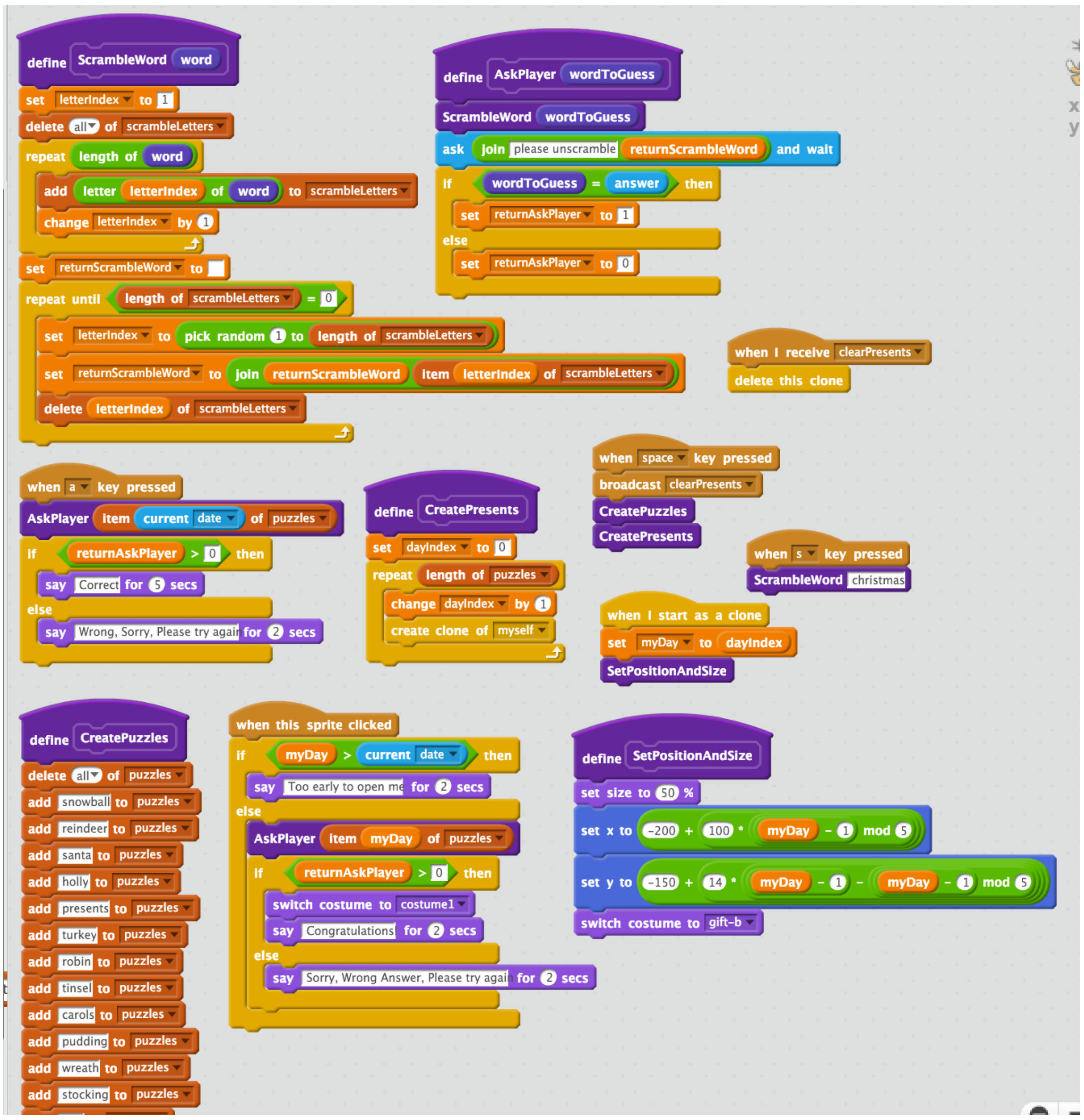
```
when this sprite clicked
if < (myDay) > (current date) > then
    say [Too early to open me] for (2) secs
else
    AskPlayer (item (myDay) of [puzzles])
    if < (returnAskPlayer) > [0] > then
        switch costume to [costume1]
        say [Congratulations!] for (2) secs
    else
        say [Sorry, Wrong Answer, Please try again] for (2) secs

define SetPositionAndSize
set size to (50) %
set x to ((-200) + ((100) * (((myDay) - (1)) mod (5))))
set y to ((-150) + ((14) * (((myDay) - (1)) - (((myDay) - (1)) mod (5)))))
switch costume to [gift-b]
```

For the sprite to look like a present, we must give it a new costume.

* For our sprite, under Costumes, delete the second costume that comes as default with the cat sprite.
* Then add another costume by picking from the library. There is a good costume that looks like a present under Things. It will be given a name like gift-b (if you pick the same blue one I did)
* Go back to the scripts for this sprite, and under the SetPositionAndSize function, we should change the costume to the present. From Looks drag switch costume to <gift-b> to the bottom of the function. Use the drop down to select the correct costume.
* Add a new event handler for receiving clicks. From Events drag When This Sprite is Clicked block into the code area.
* Clicking on a puzzle should first check that the current day is past the one for the sprite. From Control drag an if <> then else block into our event handler.
* From Operators drag a More Than > operator into our if statement.
* From Data, Drag the value of myDay into the left hand box.

* From Sensing, Drag the value of current <date> into the right hand box.
* If this condition passes, it means the player has tried to open a box too early. So simply have the sprite say this. From Looks, drag a say <hello> for <2> secs. Change the message to something like ‘Too early to open me’.
* Under the else, we should present the correct puzzle for the day to the player. So from More Blocks, drag AskPlayer block into the else condition.
* From Data, drag item <1> of puzzles into the AskPlayer parameter.
* From Data, drag myDay into the <1> of that new item selector. This means that each present will show a different puzzle when clicked.
* We should now check if the user got the answer right. This is very similar to our test function from Step 5. From Control, drag another if-else block just after the AskPlayer block.
* Use the more than operator again to check that the returnAskPlayer variable has been set to more than zero. If it has, it means the player got the question right and we can open the present.
* From Looks, drag switch costume to <costume1> into the if statement. In addition to this, from Looks drag say <hello> for <2> secs just after this and change the message to ‘Congratulations’.
* Under the else, we simply want to tell the user to try again. So from Looks, drag a say <hello> for <2> secs and change the message to ‘Sorry, wrong answer, please try again’.

CONGRATULATIONS, YOU HAVE NOW COMPLETED THE ANAGRAM ADVENT CALENDAR



```
define ScrambleWord word
set letterIndex to 1
delete all of scrambleLetters
repeat length of word
    add letter letterIndex of word to scrambleLetters
    change letterIndex by 1
set returnScrambleWord to
repeat until length of scrambleLetters = 0
    set letterIndex to pick random 1 to length of scrambleLetters
    set returnScrambleWord to join returnScrambleWord item letterIndex of scrambleLetters
    delete letterIndex of scrambleLetters
```

```
define AskPlayer wordToGuess
ScrambleWord wordToGuess
ask join please unscramble returnScrambleWord and wait
if wordToGuess = answer then
    set returnAskPlayer to 1
else
    set returnAskPlayer to 0
```

```
when I receive clearPresents
delete this clone
```

```
when a key pressed
AskPlayer item current date of puzzles
if returnAskPlayer > 0 then
    say Correct for 5 secs
else
    say Wrong, Sorry, Please try agair for 2 secs
```

```
define CreatePresents
set dayIndex to 0
repeat length of puzzles
    change dayIndex by 1
    create clone of myself
```

```
when space key pressed
broadcast clearPresents
CreatePuzzles
CreatePresents
```

```
when s key pressed
ScrambleWord christmas
```

```
when I start as a clone
set myDay to dayIndex
SetPositionAndSize
```

```
define CreatePuzzles
delete all of puzzles
add snowball to puzzles
add reindeer to puzzles
add santa to puzzles
add holly to puzzles
add presents to puzzles
add turkey to puzzles
add robin to puzzles
add tinsel to puzzles
add carols to puzzles
add pudding to puzzles
add wreath to puzzles
add stocking to puzzles
```

```
when this sprite clicked
if myDay > current date then
    say Too early to open me for 2 secs
else
    AskPlayer item myDay of puzzles
    if returnAskPlayer > 0 then
        switch costume to costume1
        say Congratulations for 2 secs
    else
        say Sorry, Wrong Answer, Please try agai for 2 secs
```

```
define SetPositionAndSize
set size to 50 %
set x to -200 + 100 * myDay - 1 mod 5
set y to -150 + 14 * myDay - 1 - myDay - 1 mod 5
switch costume to gift-b
```