Coupon Code Generation: Saving space with a simple (and insecure) hashing technique

Ben Rosenberg (bar94@cornell.edu)

March 11, 2023

1 Abstract

Have you ever shopped online?[1] What about shopping in real life?[2] If either one of these applies to you, you may be somewhat familiar with the "coupon", a thing[3] which is used at the time of purchase to reduce the amount purchased by either some fraction (for our less consumption-inclined readers, this is generally a fraction of the form $n\%$, where $0 < n < 100$ such that the application of a coupon for $n\%$-off gives a new price of $P - \frac{n}{100}P$) or a fixed amount (e.g., of the form $¤n$ wherein ¤ denotes the relevant currency sign for the purchase).

There are many types of these coupons: some apply to specific items; others to an overall purchase; still others, to multiple specific items, perhaps linking (e.g.) a specific bottle of hair product x to body wash product y. This article aims mainly to reduce the space taken up by these inevitably vast numbers of coupons, specifically those meant for use within the realm of online shopping.[4]

In this article we implement a variety of simple mathematic techniques to turn the validation of coupons from a storage-intensive task into a more computationally intensive one, with only severe drawbacks for security. We then discuss the performance of these techniques in various metrics, such as storage, runtime complexity, and security. We also discuss several simple and easily-implemented methods for improving the security of the techniques. Finally, in **Appendix A: Reference Implementation**, we provide an easy-to-follow reference implementation of these techniques, for use in commercial applications.

2 Introduction

To put it simply, we want a way to generate coupon codes (alphanumeric strings, generally of length no less than 4 characters[5]) which works within these constraints:

- The codes should be unique.
- The codes should not be easily guessable (e.g., they should not have a pattern that is easy to guess, like $000, 001, \dots$).
- The codes should be easily checked without the use of a table to store them.

One method of making sure that things[6] are correct is to use a technique known as hashing, in which you hash something. For the sake of example, in this article we use a simple hash, but this can be changed as necessary in the future.

3 Previous work

Previous work has been done in the area of the intersection of a hash and coupon codes, albeit of a slightly different type. For an example of such research please visit https://420couponcodes.com.

[1] Yes, ordering takeout counts.

[2] When was the last time you actually did this? Really makes you think.

[3] This is the scholarly-agreed-upon verbiage.

[4] There is probably room for improvement in the more physical, IRL- (in-real-life) shopping coupons as well, but due to their already negligible size as slips of paper, and the fact that they can decompose into other matter, their buildup is not perceived as being quite as harmful to the general public as is the buildup of their electronically-stored counterparts.

[5] 4 is a cool number. We can show this inductively as follows.

1. Note that 0 is a cool number. This is the case because you can't divide stuff by 0 and the graph of $\frac{1}{x}$ looks really cool when it gets close to 0. Also the Mayans invented 0 and the Mayans practiced human sacrifice and that's totally awesome. So 0 is a cool number.
2. Realize that if n is a cool number, then $n+1$ is a cool number, because $n+1$ is only one more than n. (To further this point, 1 itself is a cool number because 1 times anything is that thing again, e.g. $1 \times x = x$, which is cool because it's like the 1 is doing a magic trick where it disappears and all that's left is the number you multiplied it with. That's cool.) So $n+1$ is a cool number for all n.
3. Apply step (2) until you get to 4. Cool, right?

[6] Again, the scholarly-agreed-upon term.

4 Implementation

Consider some coupon code $C = c_1c_2c_3\dots c_n$, wherein $c_1, \dots, c_n$ are the characters which compose C. The simple hash we will use in this article is the hash which sums up the numerical representation of each of these characters, where we define the numerical representation to be a mapping η, and the hash function to be

$$h(C) = \sum_{c \in C} \eta(c).$$

We define η as follows:

$$\eta(c) = \begin{cases} \text{`A'} & 1 \\ \text{`B'} & 2 \\ \vdots & \vdots \\ \text{`Z'} & 26 \\ \text{`a'} & 27 \\ \vdots & \vdots \\ \text{`z'} & 52 \\ \text{`0'} & 53 \\ \vdots & \vdots \\ \text{`9'} & 62 \end{cases}$$

Astute readers might recognize this to be a perversion of the ASCII character code table to better fit our use case.

An example of the use of this hashing function might be the string "JJJJJJJJJJ", or `'J' × 10` for those of a more laconic persuasion.[7] Since 'J' comes out to 10 under our complex hash given above (the proof is left as an exercise for the reader), this string hashes to $10 + \dots + 10 = 10 \times 10 = 10^2 = 100$.

Now this is well and good, but how do we ensure that all strings we generate are equal under the hash? Is there even a way?[8]

Yes, indeed! This is a classic application of the balls and bins/stars and bars/balls and dividers/stars and dividers/stars and stripes/stripes and wipes/balls and wipes/wipe your balls problem, which is commonly taught in combinatorics. Say for instance that you have some 10 balls and want to put them into 3 bins. Note that you can divide these balls into groups using a grand total of $3 - 1$, also known as 2, dividers, like so:

$$\bullet \mid \bullet\bullet\bullet \mid \bullet\bullet\bullet\bullet\bullet\bullet$$

In this example, we have divided up the balls into a group of 1, a group of 3, and a group of 6. To check our work, we might add our numbers back together:

$$1 + 3 + 6 = 10$$

This technique is clearly applicable to our problem.

For the less convinced readers, it may be useful to specify that in this example, 10 is the *result* of the hashing function to which we want all of our coupons to hash, and our choice of `[1, 3, 6]` is equivalent to, under our mapping η, `ACF`.

Basically, we can split whatever number we want to add to into groups of numbers by splitting it up into $\bullet$ and then splitting those $\bullet$ into groups of $\bullet$ delineated by $|$. So we can simply choose where to put these dividers and the number of balls in each stripe or whatever corresponds to the η-defined character we use. Truly remarkable.

4.1 How do I implement this? This looks complicated and my boss says I only have until Friday to implement this and I don't want to get put on a Performance Improvement Plan please help

The solution to your quandary is that we use math. More specifically, we just generate $n - 1$ random numbers (recall that $|C| = n$) in the range $(1, |\mathrm{dom}(\eta)| = 62) \cap \mathbb{Z}$ which will correspond to the locations at which we place our dividers. We can then sort these numbers, and calculate the contents of the code from them. Let the abovementioned generated numbers be given by $\mathcal{X} = [\![x_1, x_2, \dots, x_{n-1}]\!]$, where $x_1 \le x_2 \le \dots \le x_{n-1}$, and let ₳ be the magic number to which we desire our η-map of C to add.

Since our numbers have been sorted, and since all our numbers x are within $(1, 62) \cap \mathbb{Z}$, we know that $\Delta(\mathcal{X})_{\text{₳}} = [\![x_1, x_2 - x_1, \dots, x_{n-1} - x_{n-2}, \text{₳} - x_{n-1}]\!]$ will be exclusively positive integers in $(0, 61) \cap \mathbb{Z}$. A simple transformation $x \mapsto x + 1$ gives us the domain of η^{-1}; namely, $(1, 62) \cap \mathbb{Z}$. These numbers act as the groups of $\bullet$ seen in the previous section, and correspond to the actual characters of the coupon code being generated.

[7] Other equivalently convenient ways to write the given string might be `'J' ^ 10`, or `String.make 10 'J'`, or `'J' + 'J' + 'J' + 'J' + 'J' + 'J' + 'J' + 'J' + 'J' + 'J'`, or `''.join(map(chr, [74]*10))`, or

$$f(c) := \texttt{chr(c)};$$

$$g(\ell_{\text{init}}, n, h) := \begin{cases} \ell_{\text{init}} & n = 0 \\ h(g(\ell_{\text{init}}, n-1, h)) & n > 0; \end{cases}$$

$$\lambda\left(s \mapsto s \circ f(\lfloor \pi^4 \rfloor - 2 \cdot \lceil \pi^2 \rceil - \lfloor \pi \rfloor)\right) \triangleright g(\epsilon, \lceil \pi^2 \rceil)$$

.

[8] Read on to find out!

It is important to note here that this mapping ($x \mapsto x + 1$) is an unfortunate result of the 1-indexing strategy employed in η. This strategy was only employed because the language used for the implementation (see **Appendix A: Reference Implementation**) is 1-indexed.

Denote the string concatenation operation by $\circ$. Then, more formally, we can write our coupon code C as follows:

$$C = \bigcirc_{\delta \in \Delta(\mathcal{X})} \eta^{-1}(\delta + 1) \tag{1}$$

So now, we can generate a coupon code using the following steps:

1. Generate $n-1$ random numbers $x_1, \ldots, x_{n-1}$ such that $1 \leq x_i \leq 62$
2. Convert this set of numbers into a coupon string satisfying the hash function using (1)

And we can then check that it works by:

1. Converting these characters back into numbers using η
2. Adding them up using your fingers[9]
3. Checking if the result is the equal to ₳

For readers desiring a swifter and less symbolic explanation, **Appendix A: Reference Implementation** is provided.

4.2 What is the value of ₳?

Throughout this section we have described the value ₳ as though we are familiar with it. Unfortunately, that is not the case. But we can become more familiar with it by doing some additional calculations.

Let's figure out what ₳ should be. We know that ₳ should be the sum of the character code values of $c \in C$. The sum of all the values of $\eta(c)$ for each $c \in C$ can be expressed as follows:

$$\begin{aligned}
₳ &= \sum_{c \in C} \eta(c) \\
&= \sum_{\delta \in \Delta(\mathcal{X})} (\delta + 1) \\
&= |\Delta(\mathcal{X})| + \sum_{\delta \in \Delta(\mathcal{X})} (\delta) \\
&= |C| + \big(x_1 + (x_2 - x_1) \\
&\quad + \cdots + (x_{n-1} - x_{n-2}) + (₳ - x_{n-1})\big) \\
₳ &= |C| + ₳
\end{aligned}$$

Uh... something's wrong here. It looks like our addition of 1 in the middle is throwing us off. Why were we doing that again? Oh, right, the 1-indexing. Well, it looks like we need to make our value larger, by about $|C|$.

...Okay, what should the original starting value be, though? Hm...

Okay I think it should be 62? We subtracted x_{n-1} from it right? And then that value went into η^{-1} or something, so it needs to be in the range of η. Which means that it needs to be no less than 1 and no greater than 62. So $1 \leq ₳ - x_{n-1} \leq 62$. But since $0 \leq x_{n-1} \leq 61$ we know that $1 \leq ₳ - x_{n-1} \leq 62 \Rightarrow (1, 62) \leq ₳ \leq (62, 123)$...? Okay, I'm confused.

Okay, how about this. I'll keep writing code and eventually it'll work. I'll just keep tweaking things until it looks kind of right and the values stop behaving erratically. We're using this in production, right? But we'll do some testing in UAT or something before that. Okay. If there are any real issues, someone will figure it out by the time this goes live. Yeah.

Wait, this is going live tomorrow?

5 Evaluation

Of course, this method is not exactly the most robust to attacks. Finding collisions with this hash function is easier than getting a paper accepted by SIGBOVIK, provided you know η, so it is not recommended that you use this idea. (Note that this becomes slightly harder without advance knowledge of the hashing scheme, combined with some of the methods proposed in section **6.1: Avoiding suspicion**.)[10]

That said, it does kind of work for its intended purpose, which is cutting down on space usage. If you had a lot of codes to generate, you now don't have to store them anymore, as you can check whether they are valid in real time!

To better illustrate this performance increase, we compare our method to some currently-used methods.

5.1 Comparison to alternative methods

See Figure 1.

[9] For people without fingers, Google Assistant can do addition using speech to text.

[10] Actually, even without η, this is really easy if you just know the way the hashing function works. Assuming you have access to one coupon, you already know how long the coupons are, so even if you don't know η you can generate a coupon of the given length for each possible value of $\sum_{c \in C} \eta(c)$ and you'll stumble upon the hash value eventually, allowing you to generate as many valid coupons as you want. (This is the *actual* reason that this method is so insecure.) For more information on this see Birthday Attack.

Method	Storage	Runtime	Security
Database or other similar methods	$O(n)$	$O(1)$ "amortized" (whatever that means)	Generally accepted to be "good"
Intentional Hash Collisions (this paper)	$O(1)$	Actually $O(1)$	Could be worse (probably)

Figure 1: Comparison to alternative methods

6 Improvements

6.1 Avoiding suspicion

We present several possible methods to avoid consumer suspicion:

1. Scramble your mapping η.
    - The mapping in our example given by η is very predictable, as it follows a somewhat conventional alphanumeric order. To throw potential dealscroungers off your path, consider randomizing your mapping as shown in **Appendix A: Reference Implementation**.
    - In fact, this randomization may make it significantly more difficult to check whether the coupons themselves follow any type of pattern. There are $\mathrm{nPr}(61, 61) = \frac{61!}{(61-61)!} = 61! \approx 5.076 \times 10^{83}$ different permutations of our 61-element η given above, so an attacker trying to check all possible permutations by brute force will have to check more permutations than there are atoms in the universe! (This is a good line to use on your manager when he asks how secure this is.)
2. If it is possible that others can view the source code of your checking algorithm (e.g. if, for some unfathomable reason, you make the algorithm execute client-side), make sure you minify or obscurify the code so that it becomes unreadable.[11] There are actually two bonus features to this: one, you save even *more* space, and two, your boss will have a harder time firing you when looking at your code (this is important).
3. Rate limit your coupon service?

6.2 Single-use coupons

Lastly, and most unfortunately, is the matter of preventing the multiple-time use of one-use coupons (without which extensions such as Honey might accumulate a database of coupons and notice the trend). The only way to really prevent the use of the same key multiple times is to keep track of what has and hasn't been used so far... which means saving keys rather than simply calculating them in real time and stuff.

This kind of defeats the entire purpose of our system. The best way to deal with this is likely to just turn these codes into base-62 numbers and store them in a hash table, but that wouldn't be fun. The author recommends the use of a trie-like data structure in which there are 62 children of each node (besides leaves), corresponding to adding a character to the end of a continuing word (e.g., traversing the trie starting at the root with the path 'a' → 'b' → 'c' would give the string "abc"). This might marginally cut down on repetition in the first few characters used but, of course, in the short run will be much worse than a simple set. (It is a good thought exercise, though — and will probably be useful to remember when you start interviewing again after losing your job for implementing this!) Note also that in order to gain benefits of any kind the trie should be built in real time, rather than building an entire trie wherein existence is defined via some node attribute rather than a node's existence.

7 Conclusion

This may not be the best solution to your problem. ~~This article was sponsored by Honey. Honey is a free browser add on available on Google, Opera, Firefox, Safari, if it's a browser it has Honey. Honey automatically saves you money when you checkout on sites Like Amazon. Papa John's. Kohl's. Wherever you shop it's a good chance that Honey can save you money. All you have to do when you're checking out at these major sites click that little orange button and it will scan the entire internet And find discount codes for you. It takes two clicks to install Honey. Now anytime you checkout Honey will scan the entire internet and find coupon codes for you. If there is a coupon code they will find it, and if there's not a coupon code you can rest assured that you are getting the best price possible and there literally is not one available on the internet. If you install Honey right now you can save like 50 to 100 dollars on your Christmas shopping, doing nothing. There's literally no reason not to install Honey, it takes two clicks, 10 million people use it, 100,000 5 star reviews, unless you hate money you should install Honey. If you want to install it just go to joinhoney.com/mrbeast, that's joinhoney.com/mrbeast.~~

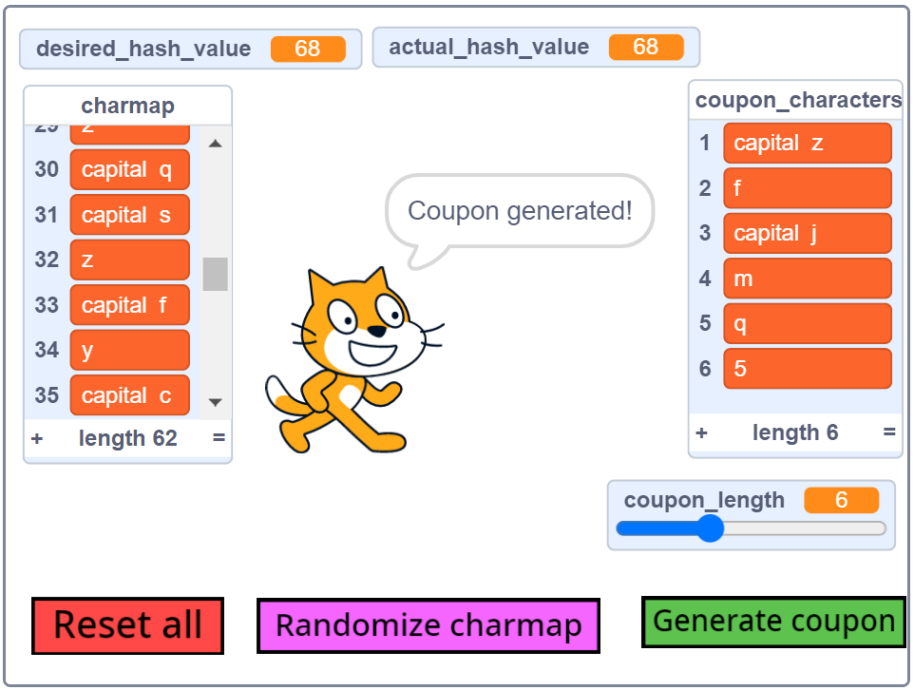
[11] That is to say, *more* unreadable than it already probably is.

References

1. In marketing, a coupon is a ticket or document that can be redeemed for a financial discount or rebate when purchasing a product. Customarily, coupons are issued by manufacturers of consumer packaged goods or by retailers, to be used in retail stores as a part of sales promotions. They are often widely distributed through mail, coupon envelopes, magazines, newspapers, the Internet (social media, email newsletter), directly from the retailer, and mobile devices such as cell phones: https://en.m.wikipedia.org/wiki/Coupon
2. Hash codes: https://420couponcodes.com
3. Honey: Honey dot com
4. Ⱥ: https://monopoly.hasbro.com/en-us
5. Mayans practiced human sacrifice: https://scholarblogs.emory.edu/gravematters/2017/02/23/human-sacrifice-in-mayan-culture/
6. Multiplying strings with integers is a thing: https://www.python.org/
7. Performance Improvement Plans: https://www.amazon.com/
8. Funny star bar bin ball thing: https://en.m.wikipedia.org/wiki/Stars_and_bars_(combinatorics)
9. There are 10^{83} atoms in the universe: https://astronomy.stackexchange.com/questions/44567/are-there-only-1083-atoms-in-the-universe
10. Google Assistant has speech to text functionality: https://support.google.com/gboard/answer/11197787
11. Birthday Attack: https://en.m.wikipedia.org/wiki/Birthday_attack#Mathematics

Appendix A: Reference Implementation

We present an implementation of the abovegiven methods below. The implementation is bundled with a GUI, for ease of use in the generation of the coupons and randomization of the character mapping. Below is a screenshot of the implemented GUI:



Another one of the limitations of the project is clearly visible in this image; namely, the lack of capital letters. Interestingly, the developers of this computational tool have decided that it is in the best interest of their users to not distinguish between capital and lowercase letters in their comparison functions, so the average unsuspecting user of the item # of thing in charmap block will be left scratching their heads for a good 15 minutes trying to figure out why searching for '`c`' gives them what they would expect for '`C`', but not vice-versa. This is the reason that we also can't directly concatenate the characters of the `coupon characters` list, as it would be misleading to clients who might interpret the use of the "capital␣" prefix as necessary.

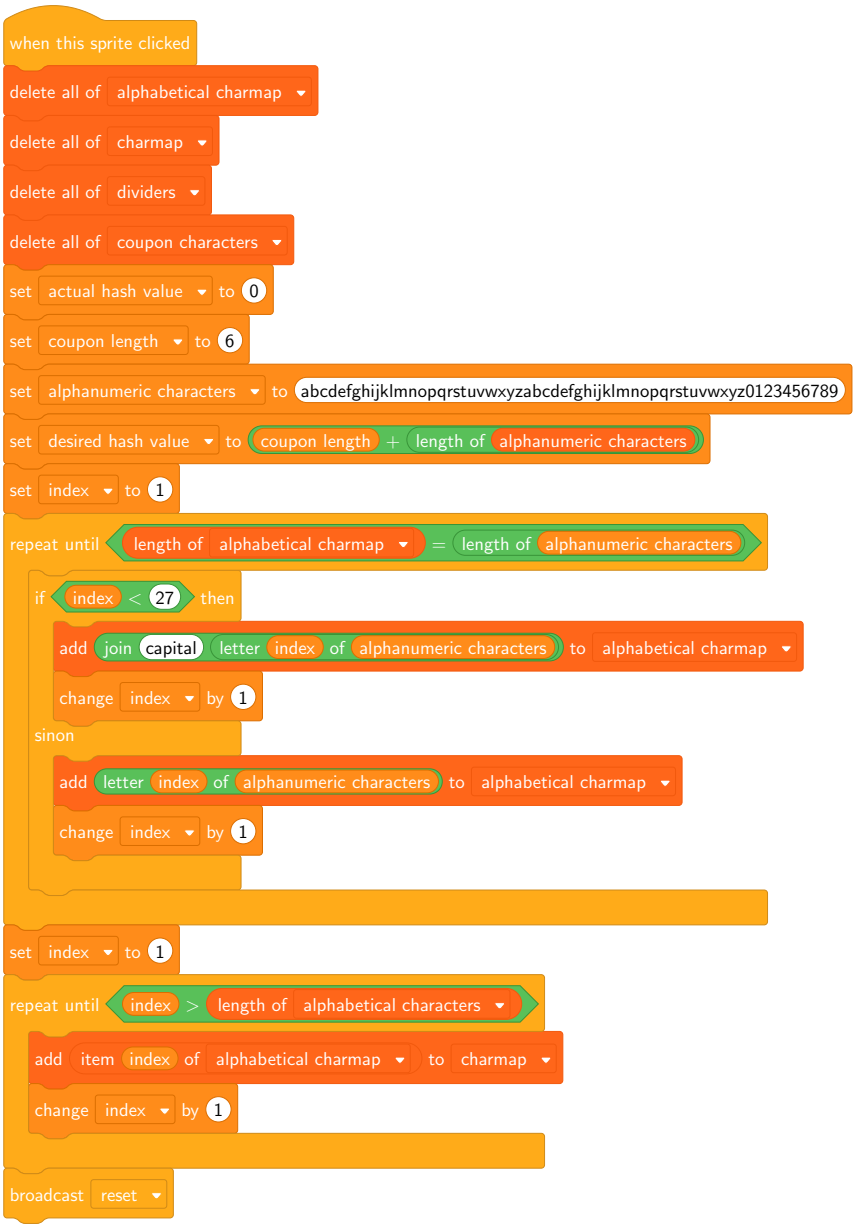
As can be seen above, the implementation is split into three different functions. In the first, the state is reset. We will call this the "Resetting". In the second, the character mapping is randomized ("Randomization"). Lastly, in the third, the coupon is generated ("Generation").

Lastly, and perhaps most importantly, we get the cat guy to say stuff when the buttons are clicked. This is referred to as "getting the cat guy to say stuff when the buttons are clicked".

Note that the following code rendering was done using the `scratch3` package in CTAN. You can read the documentation here: https://ctan.math.washington.edu/tex-archive/macros/latex/contrib/scratch3/scratch3-fr.pdf. Something to note would be that the documentation is in French. This was a slight problem for the author as the author does not know French. It is also a slight problem for the reader as the if-else block (seen below in the rendered code), instead of displaying "else", displays "sinon", which the author surmises is French for "if not". The author assures readers that despite this inconvenience, the code does in fact remain rather readable, although readers are encouraged to let the author know about their complaints regarding this matter by visiting this website: https://send-complaints.netlify.app.

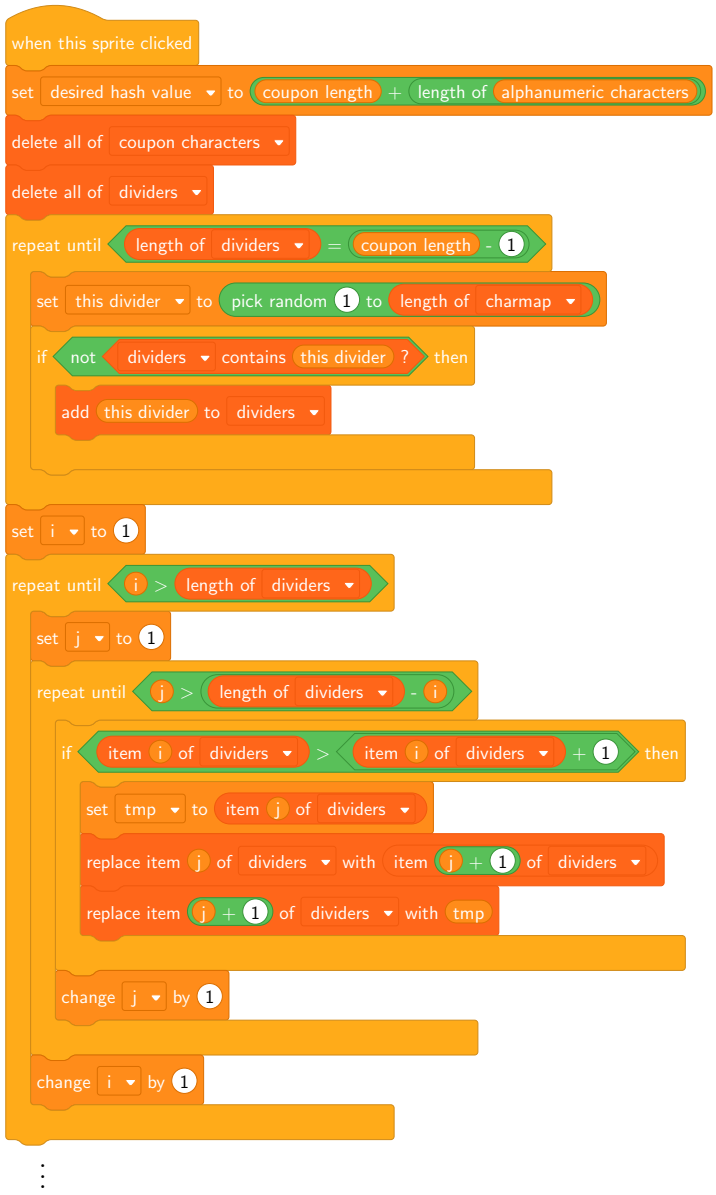
To download this code for your own use, please click here: Scratch Project (2) (1) (11).sb3

Resetting

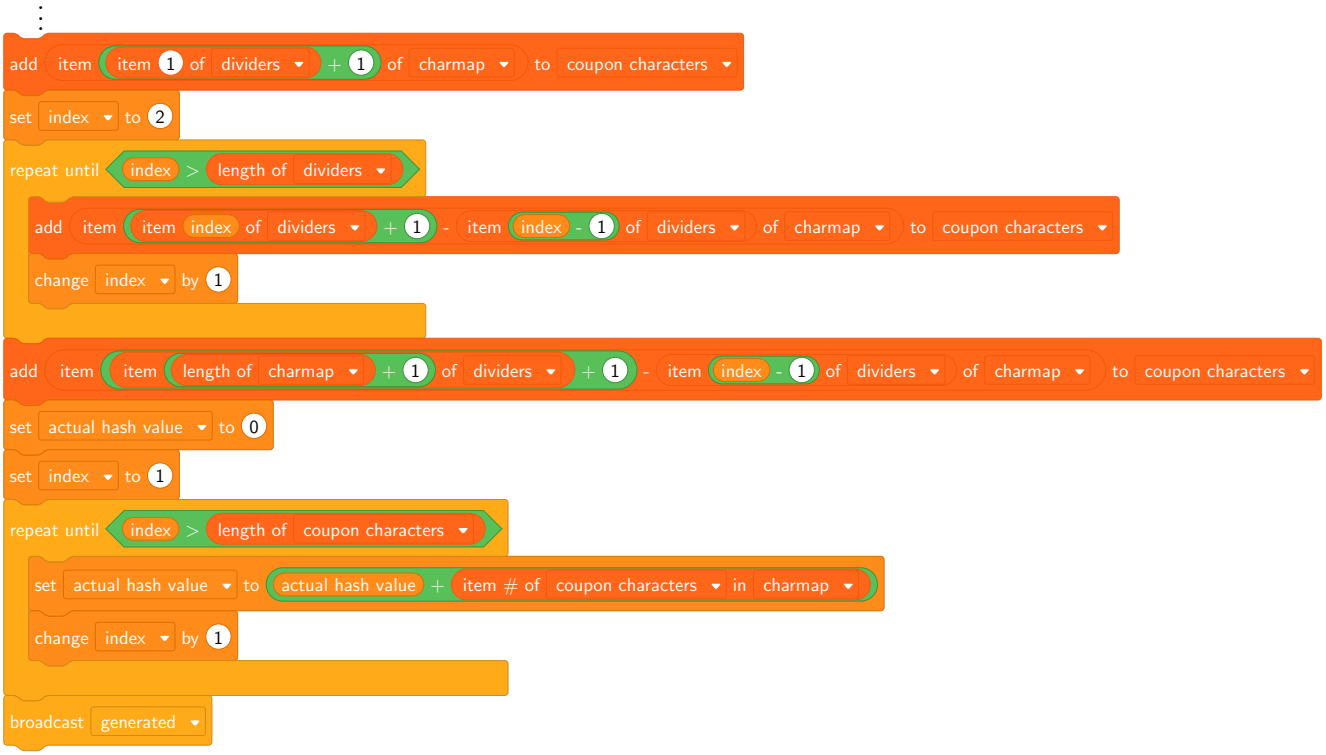

when this sprite clicked
delete all of alphabetical charmap
delete all of charmap
delete all of dividers
delete all of coupon characters
set actual hash value to 0
set coupon length to 6
set alphanumeric characters to abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz0123456789
set desired hash value to coupon length + length of alphanumeric characters
set index to 1
repeat until length of alphabetical charmap = length of alphanumeric characters
if index < 27 then
add join capital letter index of alphanumeric characters to alphabetical charmap
change index by 1
sinon
add letter index of alphanumeric characters to alphabetical charmap
change index by 1
set index to 1
repeat until index > length of alphabetical characters
add item index of alphabetical charmap to charmap
change index by 1
broadcast reset

Randomization

Generation


when this sprite clicked
set desired hash value to coupon length + length of alphanumeric characters
delete all of coupon characters
delete all of dividers
repeat until length of dividers = coupon length - 1
set this divider to pick random 1 to length of charmap
if not dividers contains this divider ? then
add this divider to dividers
set i to 1
repeat until i > length of dividers
set j to 1
repeat until j > length of dividers - i
if item i of dividers > item i of dividers + 1 then
set tmp to item j of dividers
replace item j of dividers with item j + 1 of dividers
replace item j + 1 of dividers with tmp
change j by 1
change i by 1

⋮



getting the cat guy to say stuff when the buttons are clicked