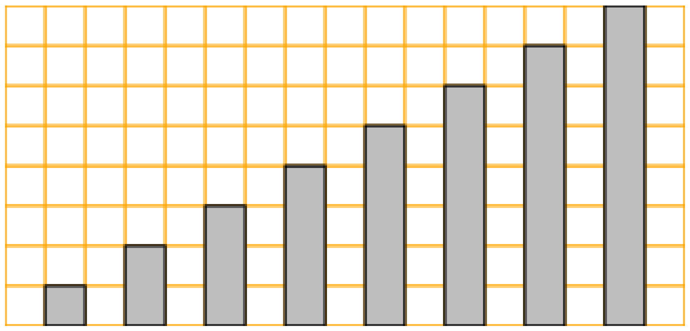
Exploration of Loops and Graphics

We consider developing code to create the following image of pillars (the dashed lines on this image are just for guidance, with this image based on a 20 × 20 grid).



We will begin with fixed geometry presuming precisely this set of eight pillars, and will eventually go back and generalize the code to an arbitrary number of pillars with an arbitrary grid size.

Basic Geometry

For the original picture, let's assume that we number the eight pillars, indexed from 0 to 7 from left to right. Fill in the following table to identify for each pillar its width (w), its height (h), and its center's x-coordinate (cx) and y-coordinate (cy).

You should detect a pattern from one pillar to the next. In the final column of the table (Δ), identify the relative change as you move from pillar "k" to "k+1".

k	0	1	2	3	4	5	6	7	Δ
w			20						
h			60						
cx			110						
cy			130						

Generalized Parameters

. The above work was assuming specifically 8 pillars and a grid size of 20. Assume that we wanted to define parameters in our source with n being the number of pillars and g being the grid size. For example, to get the original image we would define

```
n = 8    # number of pillars
g = 20   # grid size
```

If we want our script to work for any positive values n and g, we need to define our geometry relative to those parameters. Determine an expression (based on n and g) for expressing the following quantities:

Width of the entire canvas	
Height of the entire canvas	n * g
Width of the leftmost pillar	
Height of the leftmost pillar	
X-coordinate of the center of the leftmost pillar	
Y-coordinate of the center of the leftmost pillar	
Change in height from one pillar to the next (Δh)	
Change in x-coordinate from one pillar to the next (Δx)	
Change in y-coordinate from one pillar to the next (Δy)	

Implementations

There are many ways we might consider using loops to automating the process of drawing pillars on a canvas. We explore two distinct approaches here. Neither of them is really "better" than the other; they simply represent different mentalities for thinking about how to best express the necessary repetition.

You should have noticed earlier that what is different from one pillar to the next is the x- and y-coordinate of its center and its height. (The pillar widths are all the same.) So for each approach the key is how we enact the change of geometry when moving from one pillar to the next. Our high-level approaches can be described as follows:

- **Implementation 1:** We initialize variables h, cx, cy to represent the appropriate choices for the *leftmost* pillar, and then within a loop we create and configure a rectangle for a pillar, and then *update* all the variables appropriately so that they are ready for drawing the following pillar during the next pass of the loop.
- **Implementation 2:** Rather than managing the *relative* change from one pillar to the next, we can use algebra to determine a closed-form formula for $h(k)$, $cx(k)$, $cy(k)$ that express those values for pillar k, where we assume the leftmost pillar is numbered with $k = 0$.

Implementation 1

Use the answers you determined earlier in this exploration in order to replace each "?" with the appropriate expression.

```
# Implementation 1
from cs1graphics import *

n = 8 # number of pillars
g = 20 # grid size

paper = Canvas( ? , n*g)

# prepare geometry of the leftmost pillar
h = ?
cx = ?
cy = ?

for k in range(n):
    pillar = Rectangle(g, h) # set width and height
    pillar.setFillColor('gray')
    pillar.move(cx, cy) # move center to (cx,cy)
    paper.add(pillar)

    # update our variables
    h += ?
    cx += ?
    cy -= ?
```

Implementation 2

For this implementation, our code will be based on pre-determining the "closed form" expression for the parameters that vary among the pillars. For example, we will offer you the following formula for the x-coordinate of the center of the pillars.

$$cx(k) = g * (1.5 + 2 * k)$$

As a first sanity check, notice that when $k = 0$ this x-coordinate simply becomes $1.5 * k$. Note from the original picture that this is precisely where we want the center of the leftmost column as we left one full grid width empty on the left side of the canvas and then the center of this first pillar is another half grid width away from there.

However, when k is not zero, the $2 * k$ portion of the above expression represents that every time k increases, we want to move two grid widths to the right before drawing the next pillar. This is essentially the same as the Δcx term that we already deduced in earlier approaches.

Determine similarly expression for the height of pillar k and the y-coordinate of its center. Then use those calculations to complete the following implementation.

$$h(k) =$$

$$cy(k) =$$

```
from cs1graphics import *

n = 8 # number of pillars
g = 20 # grid size

paper = Canvas( ? , n*g)

for k in range(n):
    h = ?
    pillar = Rectangle(g, h)
    pillar.setFillColor('gray')
    pillar.move(g * (1.5 + 2*k), ? )
    paper.add(pillar)
```