Mathematical Reasoning with Python

GEALGEROBOPHYSICULUS PRESS

Can you count the letters in this sentence?

This is probably a silly question. Counting is usually the first mathematical skill any of us learn. It's also the first thing we will teach a computer to do.

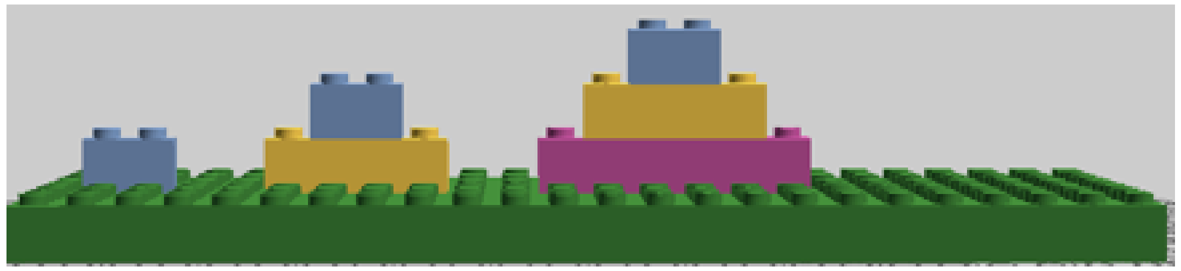
SECTION 1

How many bricks are there?

IN THIS SECTION, YOU WILL LEARN TO:

1. Apply inductive reasoning to find the total number of bricks in a tower.
2. Use Python programming to count and add things together.

I built this model built out of LEGO bricks:



The colors are different to make the different levels of each stack of bricks separate. All of the bricks used are the same type, and are like the one all the way to the left.

How many bricks are in each tower?

This is an easy enough question to answer here - we can just count them! Using a computer to answer this question would be overkill.

If there were 30, 50, or 100 rows in each tower, it would be a lot of work to count them up on our own.

Computers are really good at doing the same thing over and over again. The challenge of any programmer is making sure the computer knows exactly what it should be doing over and over again.

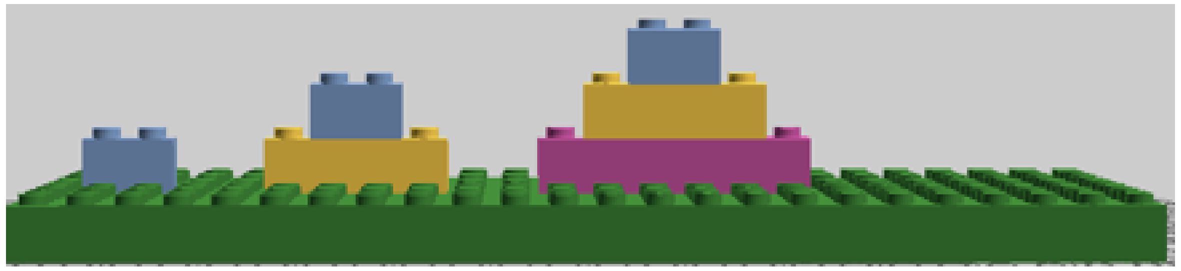
I want to show you how you might break this question down in a few ways that will help you show a computer to find the answer. Then we'll actually go about making the computer do so.

Let's organize what we know from counting:

Stack 1	1 brick
Stack 2	3 bricks
Stack 3	6 bricks

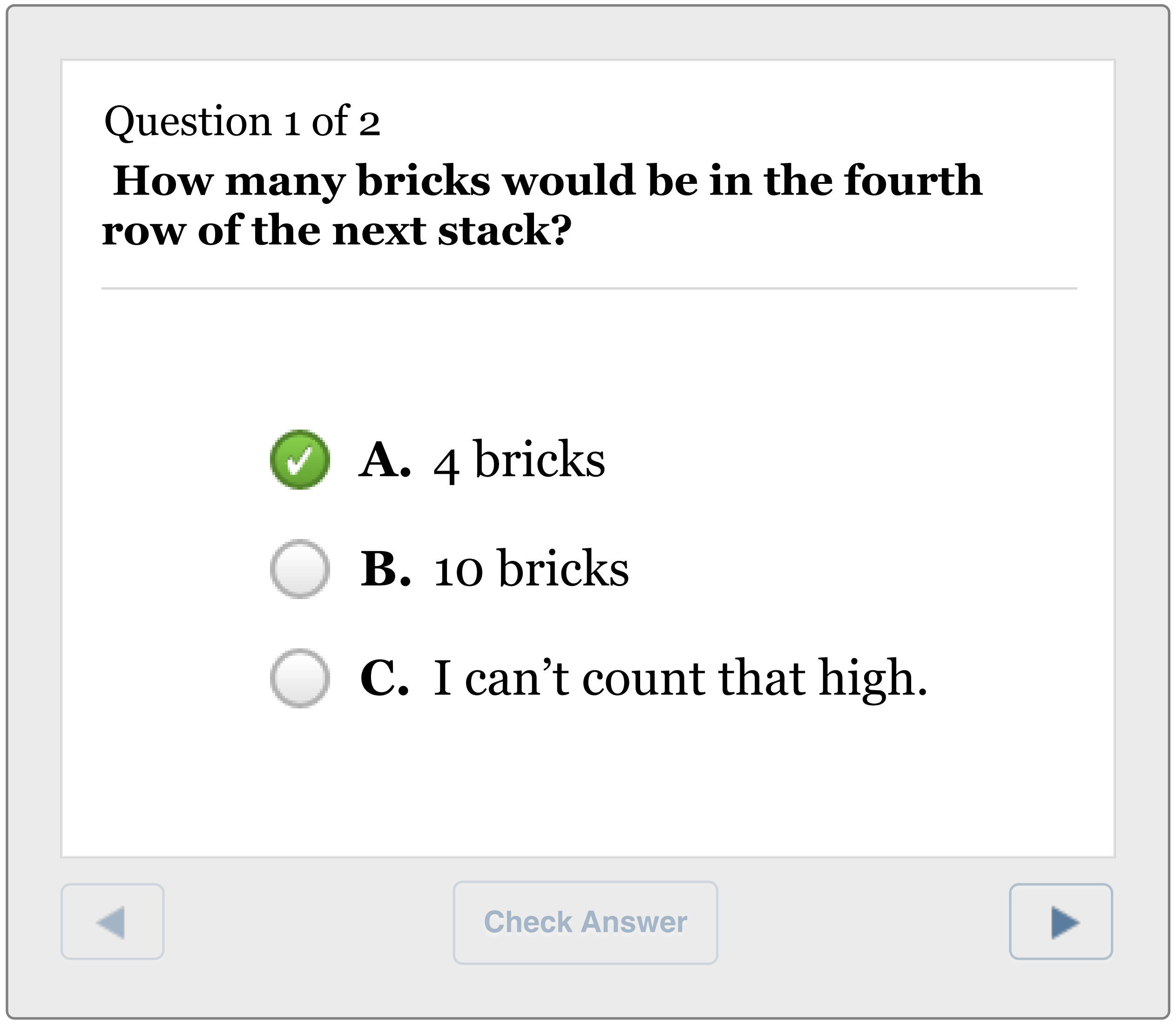
This is what we know from counting the bricks individually. In a math class, you might be asked to predict the pattern and find the next three steps, but finding those answers isn't the point yet. We need to find the simplest way to find these totals with the least amount of work possible.

There is a nice way to do this by looking at the picture again:



Can you find Stack 2 contained within Stack 3? Stack 3 is the same as adding a new row of bricks to Stack 2. You might see that Stack 2 is formed the same way: add a row of two bricks below the one brick in Stack 1.

Let's dig in a bit deeper to this idea:



Question 1 of 2

How many bricks would be in the fourth row of the next stack?

[✓ **A.** 4 bricks]

B. 10 bricks

C. I can't count that high.

The second question was a bit tricky, but it's an important thing to think about any time you are trying to find a pattern in what you see. Will the pattern continue? For how long will it continue?

If we know (or assume) the pattern continues, we can move on down the stack. If we know how many bricks are in a particular row, we can just add one to find the number of bricks in the *next* row,

Ok, it's time to fire up the Python interactive interpreter. Once you have done this, your screen should look something like this:

```
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

We are going to start by defining some variables. Variables are what a computer language (like Python) uses to store information. We will use Python to generate the numbers of bricks in each row of the stack. Suppose we are in the 15th row of the stack. We could create a variable that contains this information, and then print it out to show its value:

```
>>> current_row = 15
>>> print current_row
15
```

We can also use a variable to record the number of bricks in the current row:

```
>>> bricks_in_current_row = 15
```

We can then calculate the number of bricks in the next row by adding one to the number of bricks in the current row:

```
>>> bricks_in_current_row + 1
16
```

Notice that Python prints out what this number would be. We can store this value in a new variable that holds the number of bricks in the next row:

```
>>> bricks_in_next_row = bricks_in_current_row + 1
```

We can now print this variable as we did before to see that it contains the correct number:

```
>>> bricks_in_current_row + 1
16
```

This really shouldn't impress you, though it's just fine if it does. We still haven't made Python do anything really useful. You might have noticed that we still don't really have a procedure that the computer can follow to find

Let's go back to Stack 3 and start on the top row. We want to keep a running total of the number of bricks in each row as we look at each row of the stack. We can do this with a variable called running_total.

To start, let's say we haven't counted any bricks, so running _total=0. Type the following line into Python:

```
>>> running_total = 0
```

As we move down the stack, we can increase this variable by the number of bricks in each row.

Enough reading - it's time for you to do something.

One of the blocks of code below correctly prints out the total number of bricks in the first two rows of Stack 3. (That is, it should print out 3 at the end.) Which one is it?

Choice A:

```
>>> running_total = 0
>>> current_row = 1
>>> bricks_in_current_row = 1
>>> running_total = bricks_in_current_row
>>> bricks_in_next_row = bricks_in_current_row+1
>>> running_total = bricks_in_next_row
>>> print running_total
```

Choice B:

You don't need to type this line!

```
>>> running_total = 0
>>> current_row = 1
>>> bricks_in_current_row = 1
>>> running_total = bricks_in_current_row
>>> current_row + 1
2
>>> bricks_in_next_row = bricks_in_current_row + 1
>>> running_total = running_total + bricks_in_current_row
>>> print running_total
```

Choice C:

```
>>> running_total = 0
>>> current_row = 1
>>> bricks_in_current_row = 1
>>> running_total = bricks_in_current_row
>>> bricks_in_next_row = bricks_in_current_row + 1
>>> current_row = current_row + 1
>>> bricks_in_current_row = bricks_in_next_row
>>> running_total = running_total + bricks_in_current_row
>>> print running_total
```

Let's see if we can go step by step through what each line means. If you get lost in this, that's fine. You can move to the next page where we will play with the code altogether.

```
>>> running_total = 0
```

This says the initial value of running_total is zero, as we discussed before.

```
>>> current_row = 1
```

Here we are storing the row we are in (Row 1) in the variable named current_row.

```
>>> bricks_in_current_row = 1
```

How many bricks are in the current row? Only one. We store this value in the variable for the number of bricks in the current row.

```
>>> running_total = bricks_in_current_row
```

Since the total number of bricks so far is just 1, we can make the running total variable equal to the bricks in the current row, which is 1.

```
>>> bricks_in_next_row = bricks_in_current_row + 1
```

This stores the number of bricks in the *next* row to be one more than the current row.

```
>>> current_row = current_row + 1
```

Since we now want to move to the next row, we can keep track of this by adding one to the value of current_row.

Notice the way we can do this in Python. We are adding one to the variable current_row, and then storing this value in itself. Since current_row has a value of 1, adding 1 to current_row creates a value of 2. Using the equal sign on the left side stores this value in current_row.

```
>>> bricks_in_current_row = bricks_in_next_row
```

This also might seem a bit strange.

Why would we store the value of the variable bricks_in_next_row in the variable bricks_in_current_row?

A. Because programming is always unnecessarily complicated.

[✓ **B.**] Because we have moved to the next row. The next_row variable contains the number of bricks in the row we are now on.

Check Answer

Since we are now on row 2, the value of bricks_in_next_row contains the number of bricks in this row. We just happened to calculate this while we were on row 1.

```
>>> running_total = running_total + bricks_in_current_row
```

The value of running_total before this line is 1. There are two bricks in the current row of row 2, so this line takes the sum of 1 and 2 and stores it in running_total.

```
>>> print running_total
```

This prints the value of the variable running_total.

If your eyes glazed over during the above discussion, zone back in...NOW.

To learn what is really going on, I want you to play with the code in a particular way. Type the code below into a text editor separate from Python. This will make it easier to do the steps that are coming up.

```
running_total = 0
current_row = 1
bricks_in_current_row = 1

running_total = running_total + bricks_in_current_row
bricks_in_next_row = bricks_in_current_row + 1
current_row = current_row + 1
bricks_in_current_row = bricks_in_next_row

print running_total
```

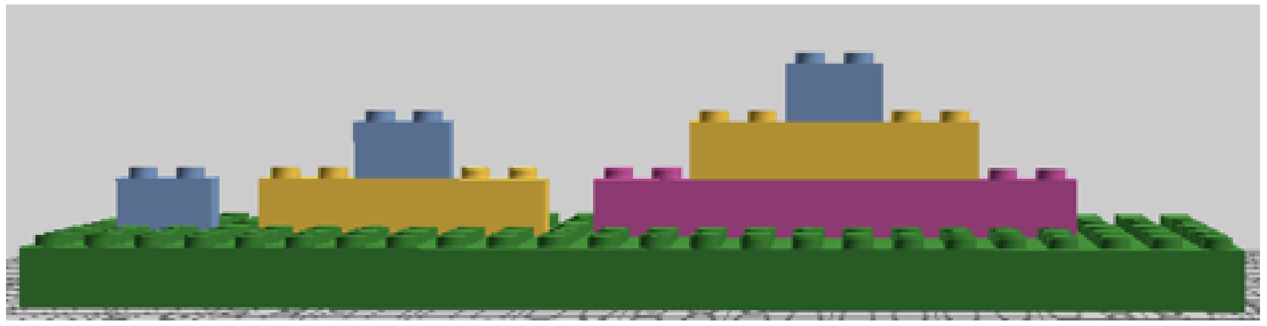
First, paste the non-highlighted code into Python and hit Enter.

Second, paste the highlighted code into Python and again hit Enter.

Paste the highlighted code again and see what happens. Do it again. And then one more time. How does the printed result relate to the number of bricks in the tower?

Do these activities before moving on!

1. Use this procedure to find the total number of bricks in a stack with ten rows.
2. Find the total number of bricks in the bottom five rows of a stack that is eight rows tall. (Do this by hand first. Adjust the program so that it gives you this value when you paste the highlighted code five times. Hint: the only code you will need to change is NOT highlighted.
3. Find the total number of bricks in the fifth stack in the series shown below. Use the code to generate your answer.



SECTION 2

Stepping it up

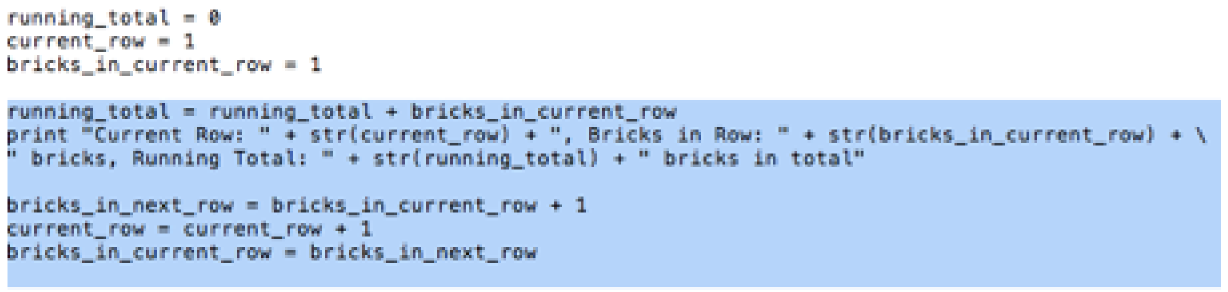
IN THIS SECTION, YOU WILL LEARN TO:

1. Use the print command to show what your program is doing in each step of a procedure.
2. Use a program structure called a loop to do repeated steps.

To solve the exercises in the last section, you needed to paste the highlighted section of code several times. These lines of code calculated the number of bricks in each row, and kept track of the total number of bricks as you moved down the stack. Still, you had to paste the code each time on your own, which would get exhausting if the stack had a large number of rows.

This may have seemed like something the computer should be able to do itself - repetition is the computer's strength, after all.

You should notice one addition in the code below.



```
running_total = 0
current_row = 1
bricks_in_current_row = 1

running_total = running_total + bricks_in_current_row
print "Current Row: " + str(current_row) + ", Bricks in Row: " + str(bricks_in_current_row) + \
" bricks, Running Total: " + str(running_total) + " bricks in total"

bricks_in_next_row = bricks_in_current_row + 1
current_row = current_row + 1
bricks_in_current_row = bricks_in_next_row
```

To make it easier for you to add this line, here is the code that you can copy and paste directly into your program in the text editor:

```
print "Current Row: " + str(current_row) + ", Bricks in Row: " + str(bricks_in_current_row) + \

" bricks, Running Total: " + str(running_total) + " bricks in total"
```

As before, paste the entire code into the Python interpreter. Now paste only the highlighted section of your code into the interpreter and see what happens. Repeat a few times.

It's nice that the program can keep us informed on what it is doing, no? Including a print command to show what is going on in each step of a procedure can be really helpful for a number of reasons:

- It makes it easier to catch mistakes in your code.
- The print command can show the programmer step by step where the answer comes from. You know how your teacher asks you to show your work when solving a problem? The print command is one way the computer can show its work.

Let's now take care of the repetition part. Python (like most programming languages) has a block of code that allows a program to do things over and over again. This block of code is called a loop.

One type of loop that we will use in this program is called a **for-loop**. This loop will do any process inside of it for a specific number of times. Let's take a look at a Python for-loop:

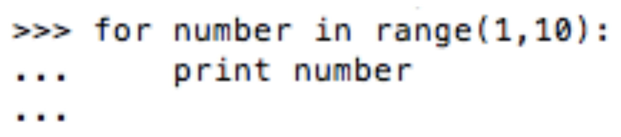
The name that comes after the **for** is always a variable. This variable keeps track of how many times the loop has repeated.

```
for number in range(1,10):
    print number
```

This means that the variable ‘number’ will have the values 1,2,3,4...8 and 9. Why not 10? Python increases ‘number’ until it is equal to the greatest integer less than the second number.

This space before the line is extremely important in Python. If you’ve ever looked at other languages, you’ll see that there are no brackets or enclosing symbols. This is because Python uses spaces like this (called indentation) to group lines of code together.

Type the code above into your Python interpreter. It should look like this:



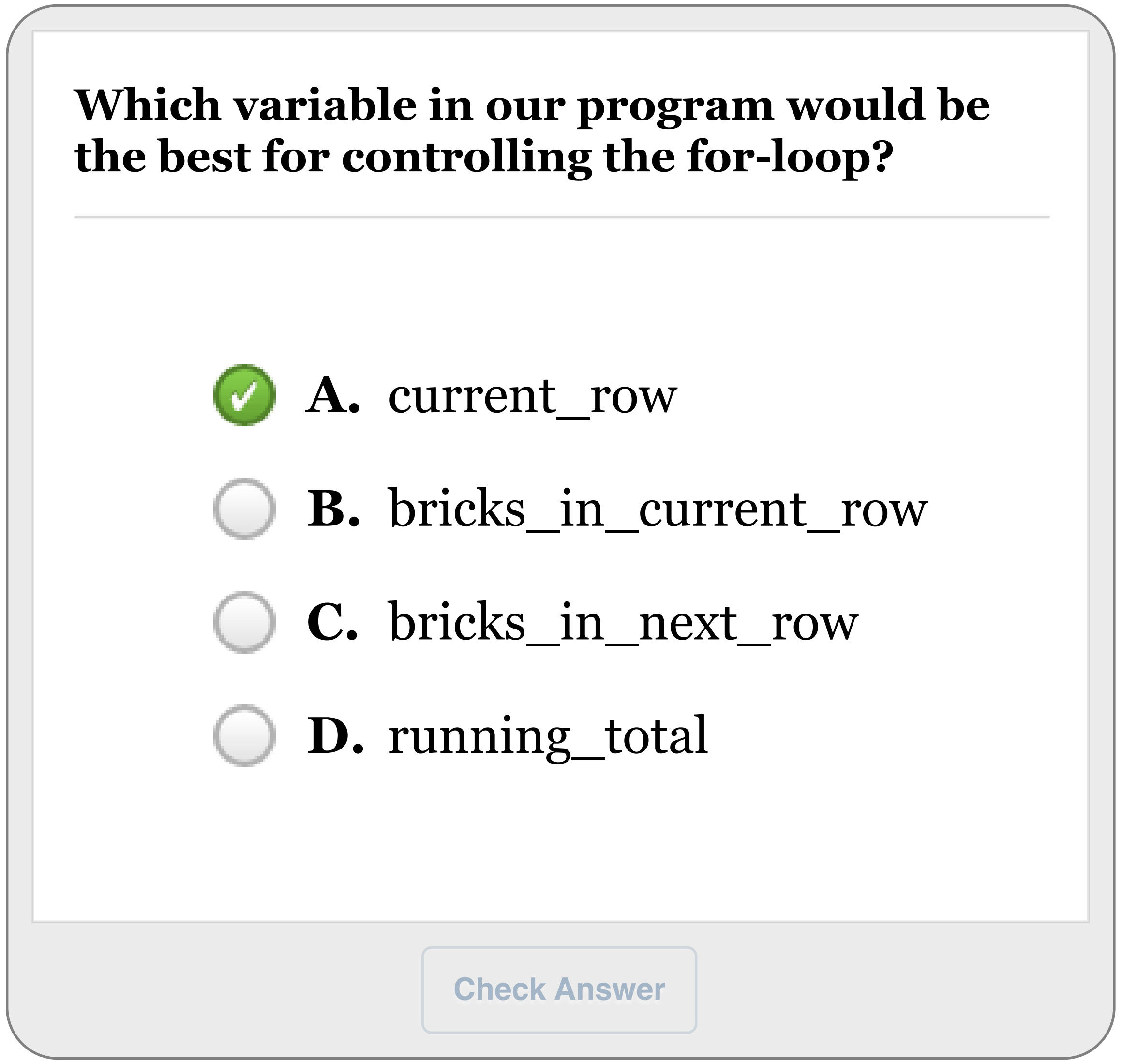
```
>>> for number in range(1,10):
...     print number
...
```

Hit Enter after the second line to run the loop. We’ve found a much faster way to count!

Do these activities before moving on!

1. Change the code so that it prints out the numbers from 1 to 500.
2. Make the code print out only the numbers 20,21,22 and 23.

Let’s now think about how to apply this to our challenge of counting up the bricks in a stack.



Which variable in our program would be the best for controlling the for-loop?

[✓ **A.** current_row]

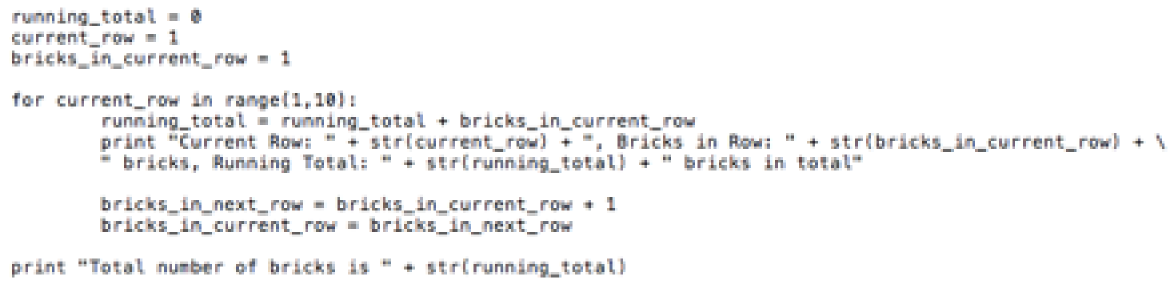
B. bricks_in_current_row

C. bricks_in_next_row

D. running_total

That current_row variable keeps track of which row in the stack we are looking at. This means that we can use it to repeat the lines of code that count up the bricks and update the running_total value for each row in the stack.

Here’s the code with those lines contained in a for loop:



```
running_total = 0
current_row = 1
bricks_in_current_row = 1

for current_row in range(1,10):
        running_total = running_total + bricks_in_current_row
        print "Current Row: " + str(current_row) + ", Bricks in Row: " + str(bricks_in_current_row) + \
        " bricks, Running Total: " + str(running_total) + " bricks in total"

        bricks_in_next_row = bricks_in_current_row + 1
        bricks_in_current_row = bricks_in_next_row

print "Total number of bricks is " + str(running_total)
```

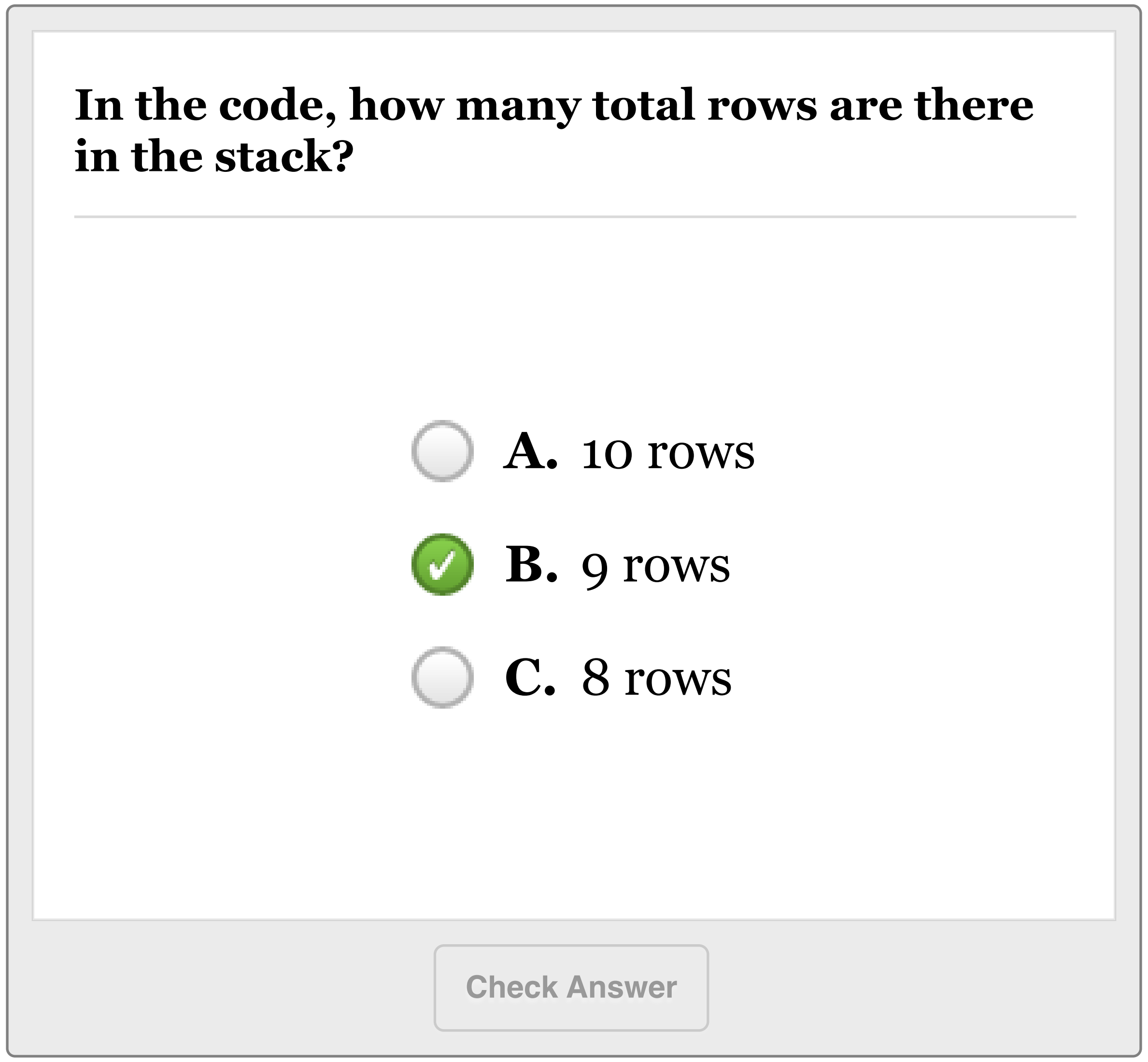
There are two key changes here, but they are pretty small.

You can see the for-loop with the lines of code that we wrote before. The line in which current_row is increased by one is gone. This is because the for-loop automatically bumps up its value once all of the lines of code within it have run.

The other change is the last line. Notice that it is not indented like the lines above it. This is because this line will only run when the for-loop has finished. It will print out the final value of running_total after iterating through all of the rows.

We’ve now completed the lines of code we need to find the number of bricks in any stack like the ones at the beginning of the chapter.

Before we get to you running this on your own computer, a quick quiz on the code above is in order.



In the code, how many total rows are there in the stack?

- **A.** 10 rows
- [✓ **B.** 9 rows]
- **C.** 8 rows

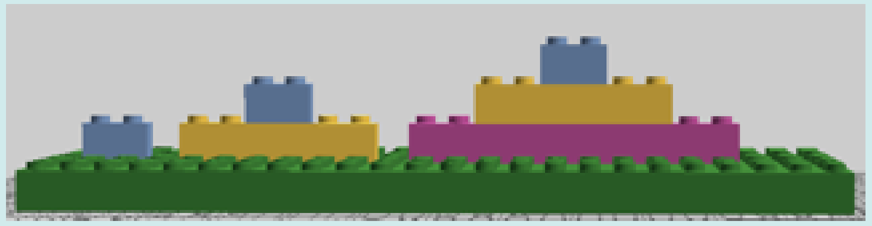
Now let’s see this in action. We’re going to run this code in a slightly different way than we’ve been doing with the Python interpreter.

Type the code into a program that will let you save a document as a plain text file. When you save the file, name the file with the extension of *.py. For example, you could name the file stack_counter.py. To run the program, go to the directory where you saved the file, and type **python** followed by the filename. For example, you might type **python stack_counter.py** for the file I named stack_counter. More information on running these programs can be found in the Appendix on installing and running Python programs.

You've done it! Now that you have a working program for finding the total number of bricks in a stack, it's time for some experimentation. See what you can do with the exercises below.

Do these activities before moving on!

1. Use the program to find the total number of bricks in a stack with 500 rows.
2. Suppose you have a stack with ten rows in it, but instead of one brick in the top row, it has **ten** bricks. How many bricks are there in this stack?
3. Find the total number of bricks in a stack with fifty rows that follows the pattern in the image below:



4. Hack the program to find the sum of the first 100 even numbers, excluding zero.