Guess My Number

Even with a large number of possibilities, there are efficient mathematical ways to home in on a number. We can program the computer to guess what number you're thinking of using bisection.

Bisection is a method for rapidly reducing the possible range for an unknown value:

- If you have an upper bound (a number which is too large) and a lower bound (a number which is too small), use the number halfway between for the next guess.

Try it yourself:

Write down a whole number between 1 and 1000, and challenge a friend to guess your value with the smallest number of yes/no questions, then change roles and guess theirs. Questions may look like "Is your number greater than 250?" and you should only make a guess when you are certain that you are correct.

For an added challenge, let the person with the secret number change their number at any point, provided the new number doesn't contradict any of their previous answers.

What is the maximum theoretical number of guesses you should need?

Go big:

If you had to guess a six-digit number (a whole number between 1 and 1,000,000), how many guesses would you need to be sure of your final answer?

Describe your method:

When you first start thinking of automating a process, a good initial step is to describe in detail what your method is. If the step requires a decision, include that in your description (eg "If the answer is yes, …, and if the answer is no, …").

You can write this out as a flow diagram, or as bullet points, or in pseudo-code (a compromise between plain English and computer code, which should be understandable by any programmer who would then be able to write it up in whatever language they wanted).

Turn over to see how to make Python guess your number …

Guess My Number *with Python*

Specification:
Make an interactive text-based Guess My Number game, where the player has a secret whole number in mind, and the computer guesses it in as few yes/no questions as possible.

Toolkit: *User Input*
You will need to get input from the user, and since users are bad at reliably giving the computer exactly the input it wants, it'll be helpful to simplify the process as much as possible. A few common approaches are shown below. All of them potentially suffer from either *false positives* (thinking a user has given a looked-for response when they haven't) or *false negatives* (thinking a user hasn't given a looked-for response when they have).

Can you see what might go wrong with each of these?

A:
```
response = input(f"Is your number larger than {guess}? ")
if response == "Yes":
  ...
```
Check to see if the user input exactly matches the word **Yes**.

B:
```
response = input(f"Is your number larger than {guess}? Y/N: ")
if response == "Y":
  ...
```
Check to see if the user input exactly matches the character **Y**.

C:
```
response = input(f"Is your number larger than {guess}? ")
if response.upper() == "Y":
  ...
```
Check to see if the user input, when capitalized, exactly matches the character **Y**.

D:
```
response = input(f"Is your number larger than {guess}? ")
if response in ["Yes", "it is", "Sure", "Yup"]:
  ...
```
Check to see if the user input can be found within a pre-defined list of positive responses.

E:
```
response = input(f"Is your number larger than {guess}? ")
if response[0] in ["y", "Y"]:
  ...
```
Check to see if the very first character of the user input can be found within a given list.

Toolkit: ***Making guesses***

By thinking about what you would do if you were the one guessing the number, you can probably see what sort of variables will be helpful to keep track of:

```
print("Hi.  Welcome to Guess My Number. ")
lower = 1
upper = 1000
print(f"Think of a whole number between {lower} and {upper}.")
count = 0
```

Initiate variables to keep track of the highest and lowest possible values of the unknown number.

How will the variables ***lower****,* ***upper*** *and* ***count*** *change as you go through the program?*

Toolkit: ***Indefinite loop***

Since the program will need to make a guess and update its values repeatedly, we'll need a loop. Since we don't know how many times to run through the loop in advance (we only stop when we find the user's number), we should use a **while** loop.

```
while upper != lower:
  ...
```

This loop uses the fact that I will update the upper and lower bounds at each stage. I know I'm done when the largest value possible is the same as the smallest.

Toolkit: ***f-strings***

In Python3, we have a more elegant way to print statements that include references to variables within the program. If we want to say "Is your number larger than 354?", we can replace the 354 with a **replacement field**, enclosed within curly braces { }, and Python will automatically evaluate the contents of that field, generate a result as a string and insert it.

```
print(f"Think of a whole number between {lower} and {upper}.")

response = input(f"Is your number larger than {guess}? Y/N: ")

print(f"Your number is {guess}. Took me {count} questions!")
```

An **f-string** is written like a normal string, with speech-marks around the contents, but preceded by an **f**. These will replace **{guess}** with the current value of the variable **guess** in the string.

Toolkit: ***Integer division***

When generating a new guess with bisection, we probably want to guess whole numbers. So what's the whole number more or less halfway between 5 and 10? A simple solution is to use integer division, but we can also convert our result (a **float**) into an integer (an **int**):

```
guess = (lower + upper) // 2

guess = int((lower + upper) / 2)
```

These both have the same effect, but if you're doing the calculation millions of times, use the first, since it never has to convert between **float** and **int** types.

A possible solution:

Below is an example piece of code which uses the ideas explained previously. It is only one possible solution, and your approach may well be better.

```
1   print("Hi.  Welcome to Guess My Number. ")
2   lower = 1
3   upper = 1000
4   print(f"Think of a whole number between {lower} and {upper}.")
5
6   count = 0
7   while upper != lower:
8     guess = (lower + upper) // 2
9     response = input(f"Is your number larger than {guess}? Y/N: ")
10    if response.upper()[0] == "Y":
11      lower = guess + 1
12    else:
13      upper = guess
14    count += 1
15
16  print(f"Your number is {upper}. Took me {count} questions!")
```

The first four lines set up the parameters, and line 6 initiates the question count.
The **while** loop from line 7 continues until the range of possible values only has a single number (eg the upper and lower bounds are the same number).
Lines 10 and 11 update the **lower** variable if the user tells us that their number is *larger* than my **guess**. That means the lowest it can be is the next integer up.
If, on the other hand, it *isn't* larger, it must be either smaller or the same, so we update **upper** to be the same as **guess** on line 13.
Finally, every time the loop runs, **count** is incremented by 1 to keep track of questions.

Debug challenge:

When I initially wrote my code, line 16 looked like this:

```
16  print(f"Your number is {guess}. Took me {count} questions!")
```

When I ran the program, thinking of the number 1000 (always a good idea to test the edge cases!) it didn't work. Can you see why?

Possible improvements / modifications:

- You could have the user decide the upper and lower bounds, or you could get creative and let them think of any number at all, forcing your program to guess increasingly large or increasingly small numbers until it finally gets and upper and lower bound to work with.
- You could adapt your program to work with floats, so that the user could think of any number whatsoever, and your program would continue to guess increasingly close until the upper and lower bounds were sufficiently close to each other.
- Can you turn it around so that the computer makes up a random number within a range (use **random.randint(1, 1000)** for instance), and you have to guess it within a certain number of questions? For example:

```
I'm thinking of a number between 1 and 1000
1. What number do you want to test? 500
My number is larger than 500.
2. What number do you want to test?
```