

Review from last week

Solution: Factoring

```
1 def factor(N):
2      $d \leftarrow 2$ 
3      $c \leftarrow N$ 
4     while  $d \leq N$  do
5          $(q, r) \leftarrow \text{divide}(c, d)$ 
6         if  $r = 0$  then
7             print( $d$ )
8              $c \leftarrow q$ 
9         else
10             $d \leftarrow \text{add}(d, 1)$ 
11 end def
```

Divide by 2 until you can't divide by 2 anymore. Then divide by 3 until you can't divide by 3 anymore, etc.

Binary search

Number guessing game

The judge picks a number from 1 to 100. The guesser has to guess it. After each guess, the judge will tell you whether your guess was correct, too high, or too low.

What's a good strategy for guessing the number quickly?

Quick review of lists

- Python list datastructure
- Also often called "array"
- Combine multiple pieces of data together.
- Access elements with numeric indexing.
 - Zero-based counting.
 - First element is at index 0, second element is at index 1, third element is at index 2, and so on.

Example:

```
animals = ["cat", "dog", "snake"]
```

```
animals[0] == "cat"
animals[2] == "snake"
```

```
# Length
len(animals) == 3
```

```
# Due to zero-based indexing, the last element is:
animals[len(animals) - 1] == "snake"
```

Linear array search

- 1 *Parameters:* Take an array of numbers and a target element 'n' to look for.
- 2 *Return:* The index where we found the target.

```

3 def linear_find(arr, target):
4      $i \leftarrow 0$ 
5     result  $\leftarrow$  None
6     while  $i < \text{len}(\text{arr})$  do
7         if arr[ $i$ ] = target then
8             result  $\leftarrow i$ 
9             stop looping because we found target
10        else if arr[ $i$ ] > target then
11            If the array is sorted, stop looping because we've gone past where target could be, to
            numbers bigger than target.
12        end if
13         $i \leftarrow i + 1$ 
14    end while
15 end def

```

If the array is sorted, we can stop looking after we find a number larger than n , even if we didn't find n . And we can return some special value like None or -1 to indicate failure.

Binary array search

```
1 Parameters: Take an array of numbers and a target element to look for.
2 Return: The index where we found the target.
3 def binary_find(arr, target):
4      $L \leftarrow 0$ . Left boundary
5      $R \leftarrow \text{len}(\text{arr}) - 1$ . Right boundary
6     while  $L \leq R$  do
7         middle  $\leftarrow \text{floor}(\frac{L+R}{2})$ 
8         if arr[middle] < target then
9              $L \leftarrow \text{middle} + 1$ 
10        else if arr[middle] > target then
11             $R \leftarrow \text{middle} - 1$ 
12        else, they're equal, we've found target
13            return middle
14        end if
15    end while
16    return None
17 end def
```

Demo on the board with a list of just numbers 1 through 10, drawing L and R pointers. Crossing out parts of the list beyond L or beyond R.