Searching Algorithms

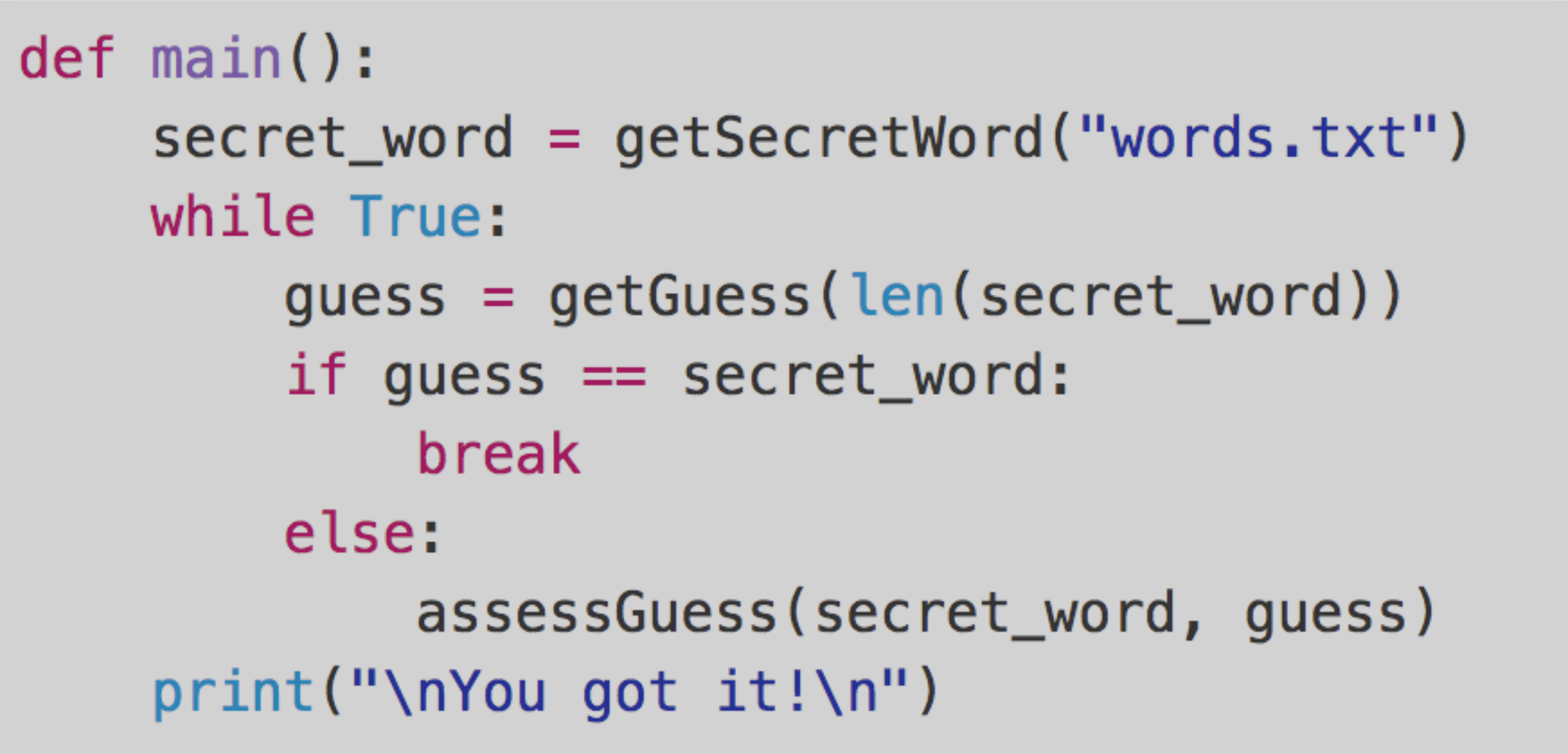
Announcements

- Lab 7 is due Saturday, March 25 (not March 28)
- Quiz 4 is on Friday, study guide has been posted
- You will hear back from me about TDD today, if you haven’t already

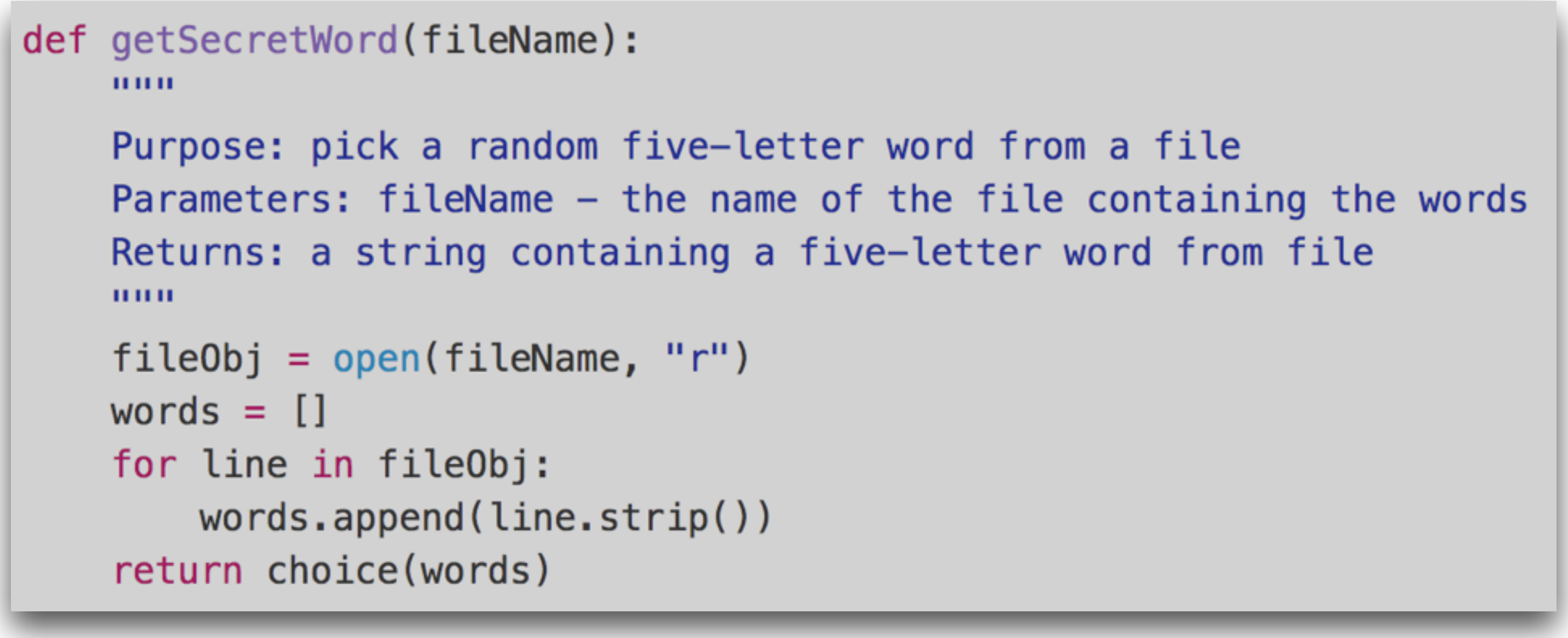
Today’s Plan

- mastermind.py (searching)
- Linear search
- Binary search

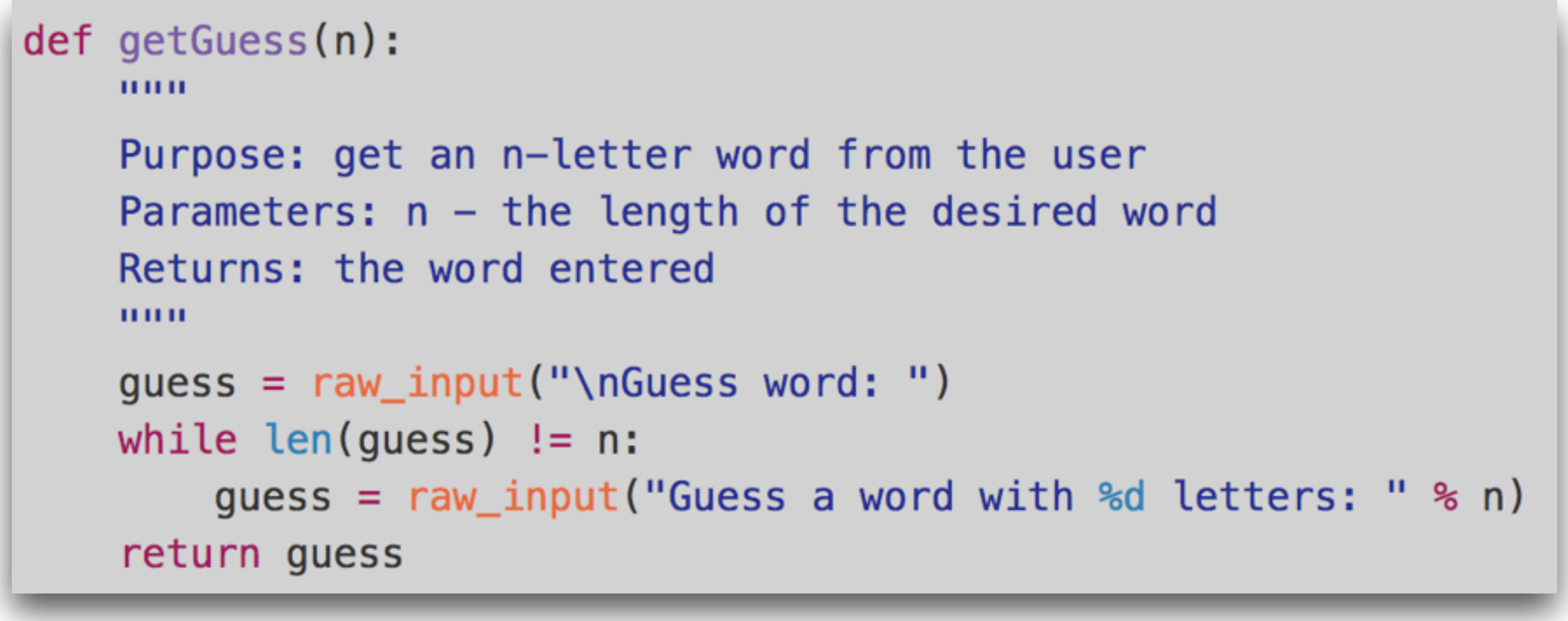
mastermind.py - TDD



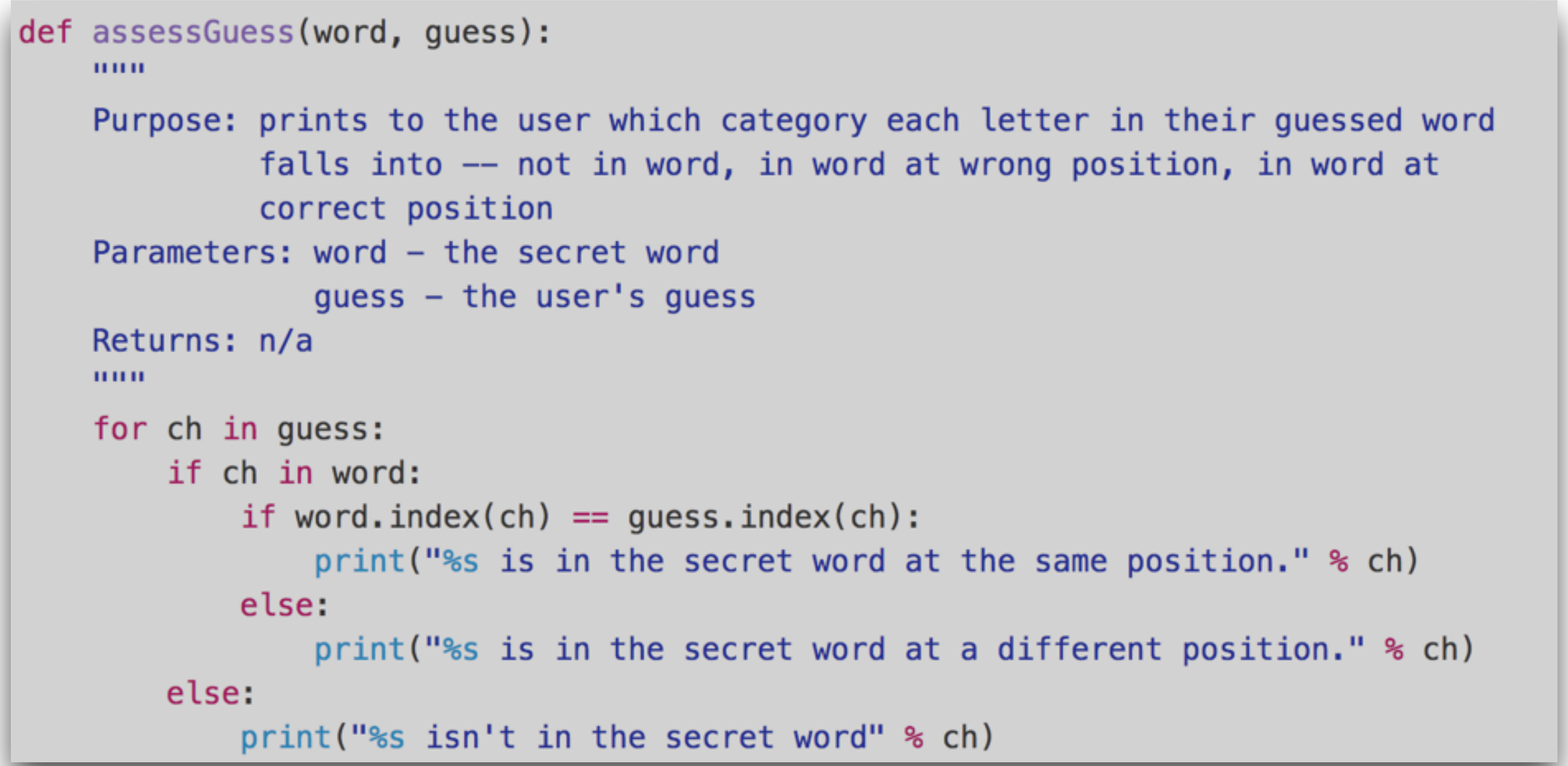
```python
def main():
    secret_word = getSecretWord("words.txt")
    while True:
        guess = getGuess(len(secret_word))
        if guess == secret_word:
            break
        else:
            assessGuess(secret_word, guess)
    print("\nYou got it!\n")
```



```python
def getSecretWord(fileName):
    """
    Purpose: pick a random five-letter word from a file
    Parameters: fileName - the name of the file containing the words
    Returns: a string containing a five-letter word from file
    """
    fileObj = open(fileName, "r")
    words = []
    for line in fileObj:
        words.append(line.strip())
    return choice(words)
```



```python
def getGuess(n):
    """
    Purpose: get an n-letter word from the user
    Parameters: n - the length of the desired word
    Returns: the word entered
    """
    guess = raw_input("\nGuess word: ")
    while len(guess) != n:
        guess = raw_input("Guess a word with %d letters: " % n)
    return guess
```



```
def assessGuess(word, guess):
    """
    Purpose: prints to the user which category each letter in their guessed word
             falls into -- not in word, in word at wrong position, in word at
             correct position
    Parameters: word - the secret word
                guess - the user's guess
    Returns: n/a
    """
    for ch in guess:
        if ch in word:
            if word.index(ch) == guess.index(ch):
                print("%s is in the secret word at the same position." % ch)
            else:
                print("%s is in the secret word at a different position." % ch)
        else:
            print("%s isn't in the secret word" % ch)
```

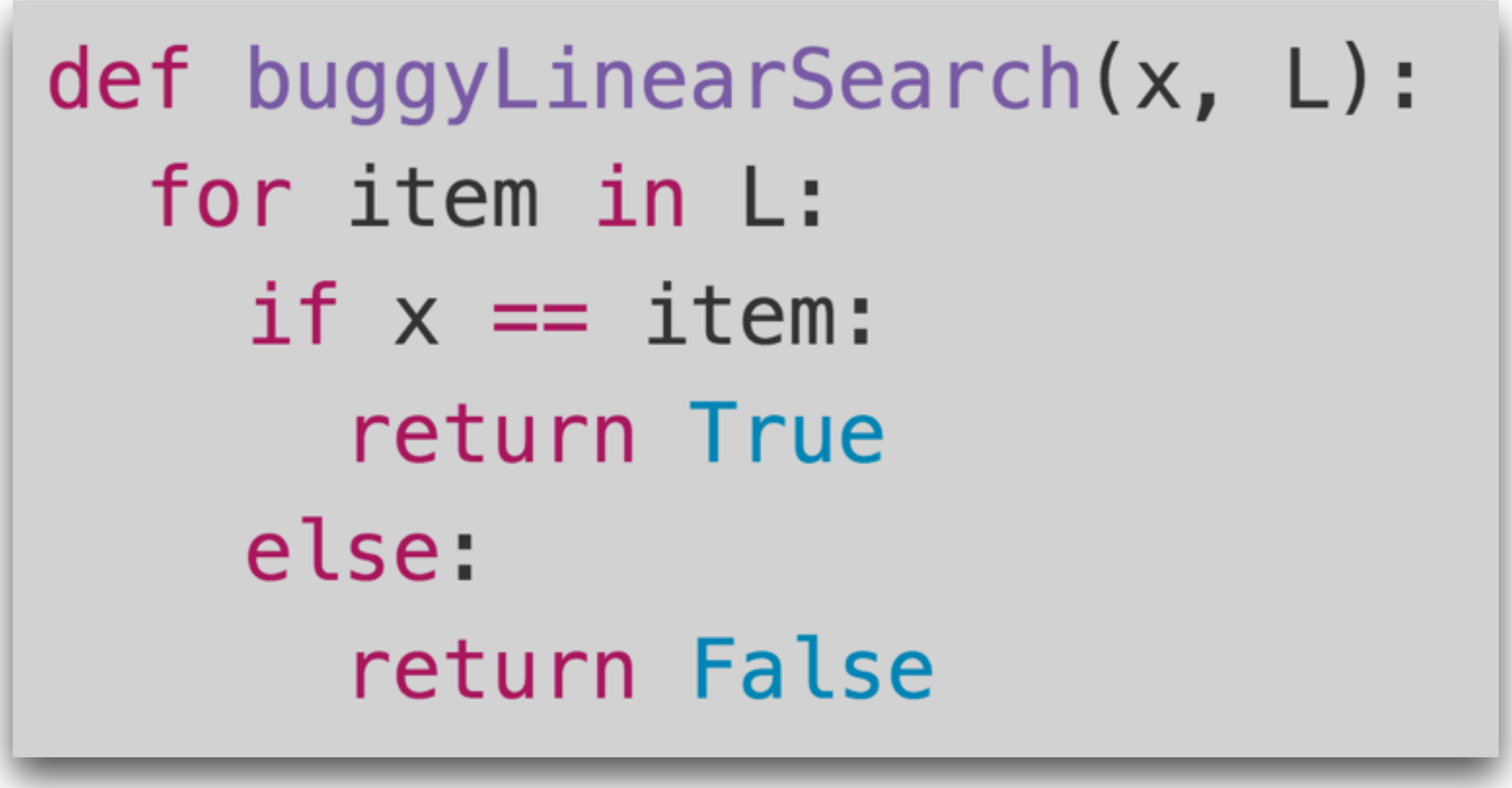
Can we write a function that acts like ***`in`*** operator?

Linear search

- Go through the items in a list/sequence, ***L***, one-by-one comparing with the searched-for value, ***x***

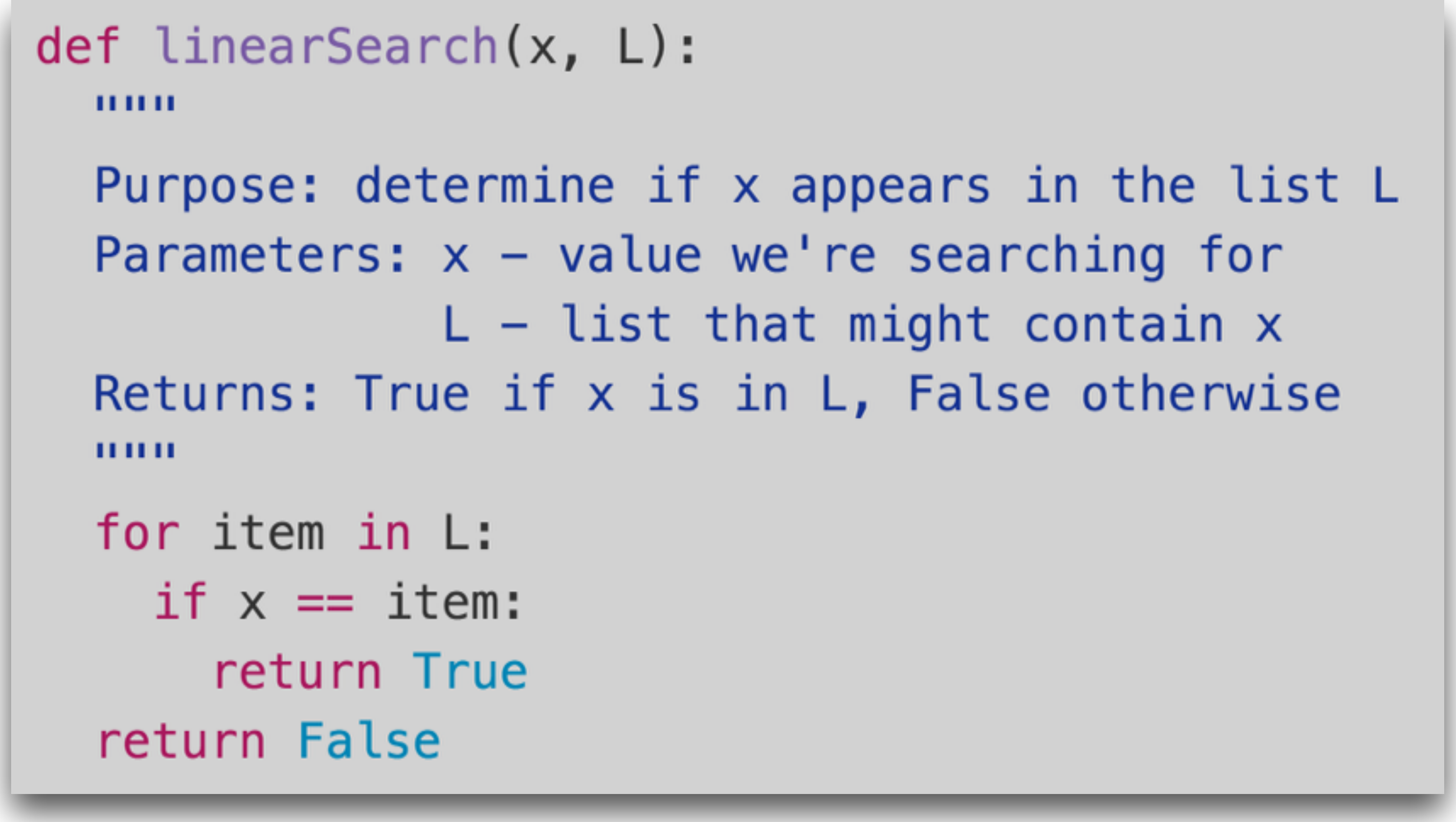
- If ***x*** isn't there, we have to check every item in ***L*** before we can be sure.

Why won’t this work?



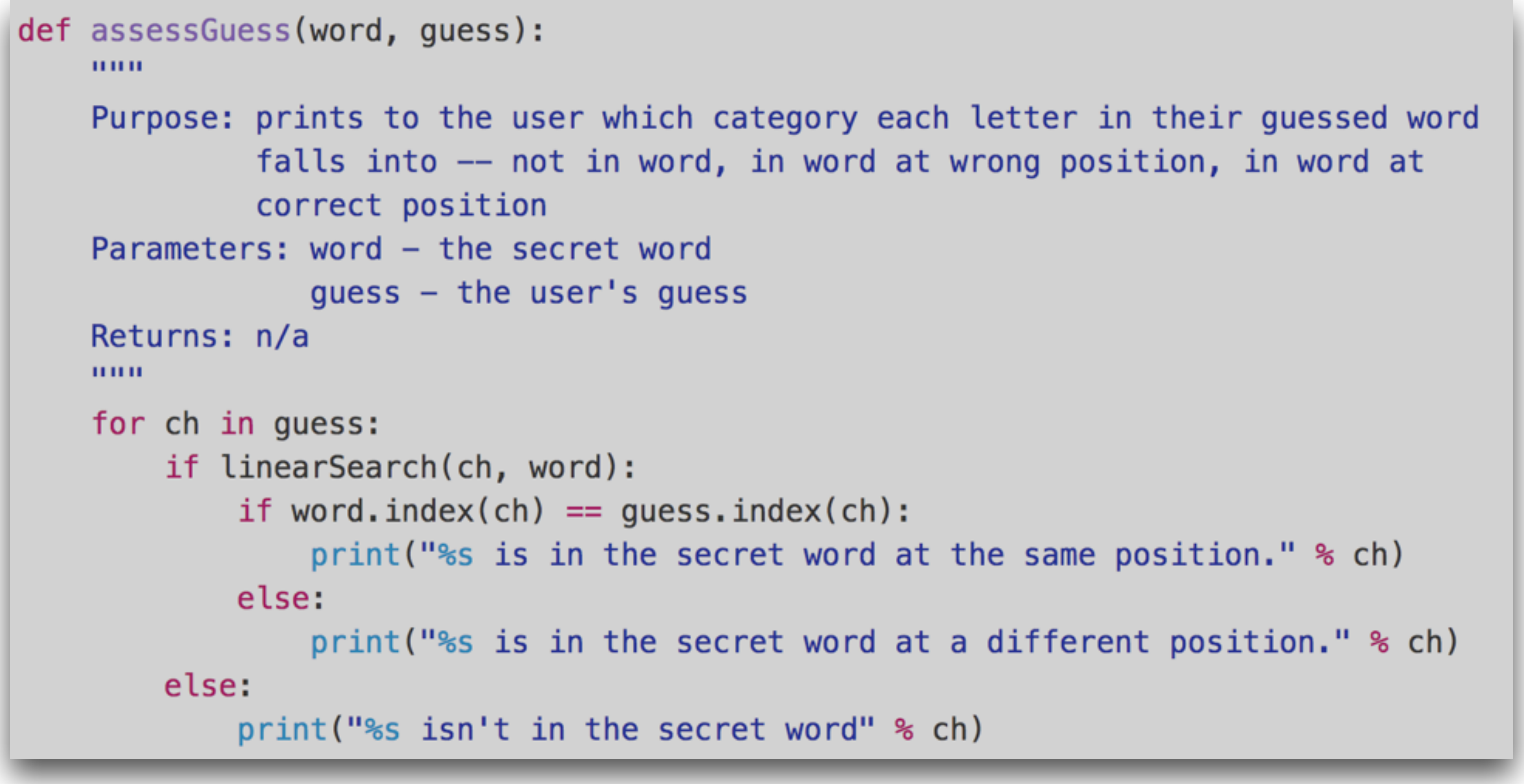
```python
def buggyLinearSearch(x, L):
  for item in L:
    if x == item:
      return True
    else:
      return False
```

Linear search



```python
def linearSearch(x, L):
  """
  Purpose: determine if x appears in the list L
  Parameters: x - value we're searching for
              L - list that might contain x
  Returns: True if x is in L, False otherwise
  """
  for item in L:
    if x == item:
      return True
  return False
```

assessGuess w/ search



```
def assessGuess(word, guess):
    """
    Purpose: prints to the user which category each letter in their guessed word
             falls into -- not in word, in word at wrong position, in word at
             correct position
    Parameters: word - the secret word
                guess - the user's guess
    Returns: n/a
    """
    for ch in guess:
        if linearSearch(ch, word):
            if word.index(ch) == guess.index(ch):
                print("%s is in the secret word at the same position." % ch)
            else:
                print("%s is in the secret word at a different position." % ch)
        else:
            print("%s isn't in the secret word" % ch)
```

Linear search

- Task: find value in a list

- Algorithm: compare ***`x`*** with each item in ***`L`*** one-by-one. If there's an item that's equal to ***`x`***, return ***`True`***. If we get to the end of ***`L`*** without finding such a value, return ***`False`***

- So the **run time** of the algorithm is proportional to the length of the list.

Can we do better?

- Normally, no. But if ***L*** is **sorted**, then we do have a faster algorithm. (We'll talk on Wednesday about how we measure the speed of an algorithm.)

- Remember the 'guess my number' program?

- Idea: each time we compare ***x*** with an item in ***L***, we either have found ***x*** or we can cut the number of candidates in half.

Binary search

- Task: find value in a list (same as linear search)
- Condition: list must already be in sorted order

Binary search

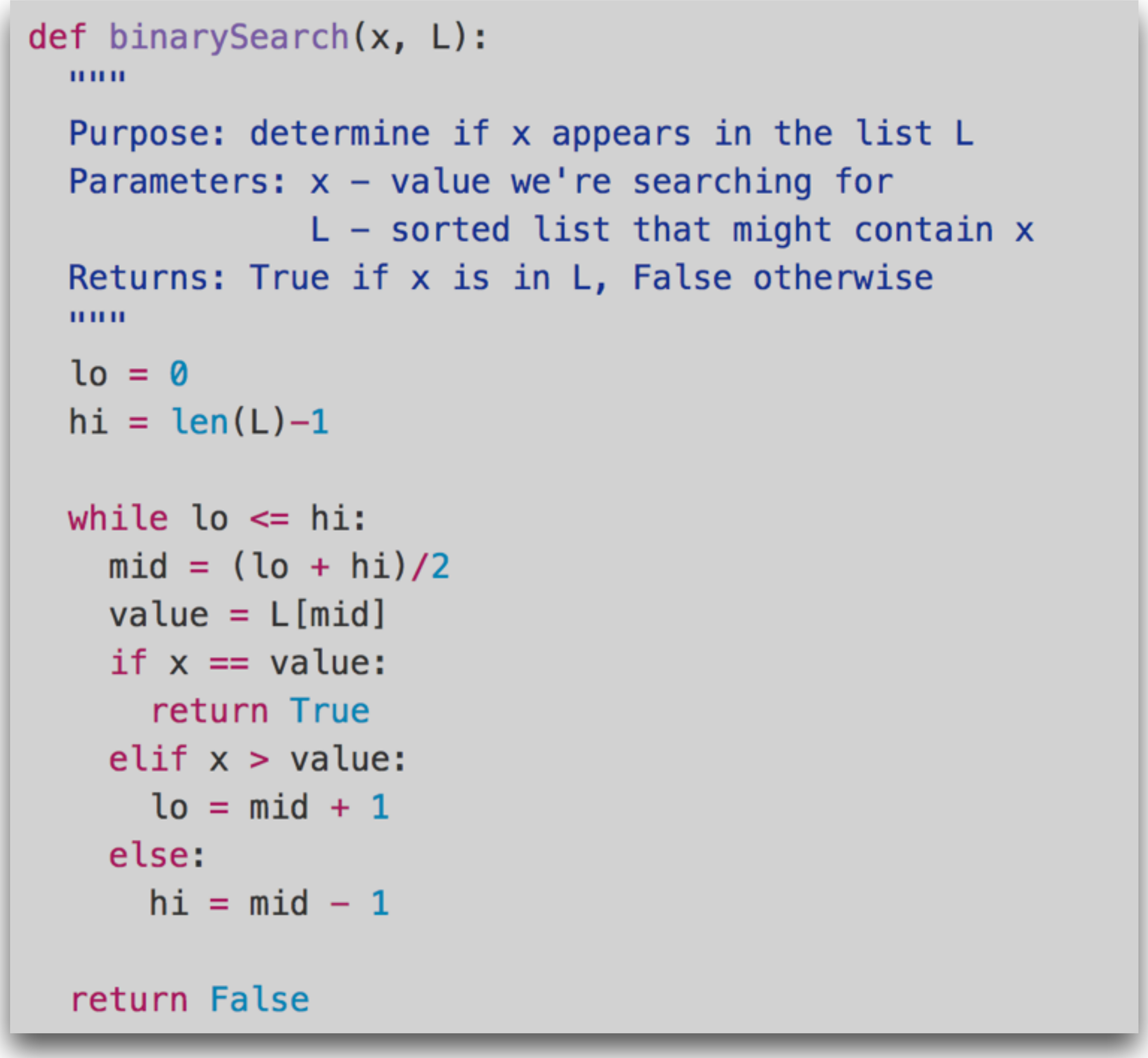
- Algorithm:

  1. Keep track of the smallest possible index where ***`x`*** might be (***`lo`***) as well as the highest possible index where the value might be (***`hi`***). Initially ***`lo = 0`*** and ***`hi = len(L) - 1`***

  2. Calculate the index midway between ***`lo`*** and ***`hi`*** (***`mid`***).

  3. Examine the item at index ***`mid`***. If it's the same as ***`x`***, return ***`True`***. Else if it's less than ***`x`***, set ***`lo = mid + 1`*** and return to step 2. Else it's bigger than ***`x`***, set ***`hi = mid - 1`*** and return to step 2.

  4. If ***`lo`*** ever becomes bigger than ***`hi`***, return ***`False`***



```
def binarySearch(x, L):
  """
  Purpose: determine if x appears in the list L
  Parameters: x - value we're searching for
              L - sorted list that might contain x
  Returns: True if x is in L, False otherwise
  """
  lo = 0
  hi = len(L)-1

  while lo <= hi:
    mid = (lo + hi)/2
    value = L[mid]
    if x == value:
      return True
    elif x > value:
      lo = mid + 1
    else:
      hi = mid - 1

  return False
```