Assignment: You are given a list of numbers, and a separate number x. Develop an algorithm to determine whether there exists two numbers in the list that sum to x.

1. Write a fruitful function named `findOperandsList1` that accepts two arguments — a list of integers, and a separate integer — and that returns true if there are two different numbers in the given list that add up to the given integer, false otherwise.

   For this function, use a list-analysis approach. You are not permitted to create any new lists nor to use a dictionary.

2. Now write a fruitful function named `findOperandsDict` having the same two arguments as before and that solves the same problem. For this function, use a dictionary-based approach. Your solution should be $O(n)$.

3. Now write a fruitful function named `findOperandsList2` having the same two arguments as before and that solves the same problem. For this function, **assume that you are only working with non-negative integers** and then use a separate-list-based approach (i.e., you can use a new separate list, but do not use a dictionary) based on a similar idea for your dictionary-based approach.

4. In the online reflection on Lyceum, or as a separate attached PDF, provide answers to the following:
   (a) Briefly describe your algorithm that uses list analysis (not using any additional lists). Use a high-level discussion; do not tell me line-by-line what your code does.
   (b) Briefly describe your algorithm that uses a dictionary.
   (c) Briefly describe your algorithm that uses a separate list.
   (d) Give the big-Oh analysis of your list-analysis algorithm, and provide justification for that analysis.
   (e) Give the big-Oh analysis of your dictionary-based algorithm, and provide justification for that analysis.
   (f) Give the big-Oh analysis of your additional-list-based algorithm, and provide justification for that analysis.
   (g) Consider the following code, which looks for a sum that cannot be in the randomly-generated list. Execute first for `list_size` of 10,000 and report the timings. Then execute for `list_size` of 100,000 and report the timings. What do you notice about the timings?

```
random.seed(8675309)

max_val   = 10000
list_size = 10000  # then try 100000
look_for  = max_val + max_val + 1

l = [random.randint(1, max_val) for i in range(list_size)]

start = time.perf_counter()
result = findOperandsList1(l, look_for)
end = time.perf_counter()
print(f"list1: result found in end-start secs")

start = time.perf_counter()
result = findOperandsDict(l, look_for)
end = time.perf_counter()
print(f"dict:  result found in end-start secs")

start = time.perf_counter()
result = findOperandsList2(l, look_for)
end = time.perf_counter()
print(f"list2: result found in end-start secs")
```

   (h) From the previous, you should have seen that timing-wise, your (third) additional-list-based approach executes in time similar to that using a dictionary. Explain why, in general, the dictionary approach might still be preferred. (Hint: Consider two things — the inputs you are testing on, as well as the space that is required when your algorithms actually do find two entries that sum to the given value.)