EXERCISES

DEDUPLICATE A LIST

Write a function that takes a list as input and returns a new list that contains all the elements of the first list minus all the duplicates.

SOLUTION

```
def dedupe_v1(x):
  y = []
  for i in x:
    if i not in y:
      y.append(i)
  return y
```

ROCK-PAPER-SCISSORS GAME

Make a two-player Rock-Paper-Scissors game.

Print the outcome of the game in the screen.

SOLUTION

```
import sys

user1 = input("What's your name?")
user2 = input("And your name?")
user1_answer = input("%s,do you want to choose rock, paper or scissors?" % user1)
user2_answer = input("%s,do you want to choose rock, paper or scissors?" % user2)

def compare(u1,u2):
    if u1 == u2:
        return("It's a tie!")
    elif u1 == 'rock':
        if u2 == 'scissors':
            return("Rock wins!")
        else:
            return("Paper wins!")
    elif u1 =='scissors':
        if u2 == 'paper':
            return("Scissors win!")
        else:
            return("Rock wins!")
    elif u1 == 'paper':
        if u2 == 'rock':
            return("Paper wins!")
        else:
            return("Scissors win!")
    else:
        return("Invalid input! You have not entered rock, paper or scissors, try again.")
        sys.exit()

print(compare(user1_answer,user2_answer))
```

ELEMENT SEARCH

- Write a function that takes an ordered list of numbers (from smallest to largest) and another number as input.
- The function decides whether or not the given number is inside the list and prints the appropriate boolean. It prints 1 if the number is in the list and 0 if it isn’t.
- Use binary search!

SOLUTION

Binary Search

- Search a sorted array by repeatedly dividing the search interval in half.
- Begin with an interval covering the whole array.
- If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

SOLUTION

```
def find(ordered_list, element_to_find):
  start_index = 1
  end_index = len(ordered_list) - 1

  while True:
    middle_index = (end_index - start_index) / 2

    if middle_index < start_index or middle_index > end_index or middle_index < 0:
      return False

    middle_element = ordered_list[middle_index]
    if middle_element == element_to_find:
      return True
    elif middle_element < element_to_find:
      end_index = middle_index
    else:
      start_index = middle_index
```