In [39]:

```
def strictly_increasing(lst):
    """
    function that takes in a list of values and creates a list
    consisting of strictly increasing values.
    Args:
        Lst (list): list of ints
    Returns:
        A list of ints in strictly increasing order.
    >>> strictly_increasing([1, 3, 2, 4, 5, 8, 7, 6, 9])
    [1, 3, 4, 5, 8, 9]
    """
    output = []
    for num in lst:
        if len(output) == 0:
            output.append(num)
        elif num > output[-1]:
            output.append(num)
    return output
print(strictly_increasing([1, 3, 2, 4, 5, 8, 7, 6, 9]))
```

```
[1, 3, 4, 5, 8, 9]
```

In [38]:

```
def all_palindromes(lst_of_words):
    """
    function that takes in a list of strings and determines if
    every element is a palindrome or not.
    args:
        lst_of_words (list): A list of strings.
    returns:
        True if every element is a palindrome, otherwise False.
    >>> all_palindromes(['a',''])
    True
    >>> all_palindromes(['ab', 'aaa'])
    False
    """
    for word in lst_of_words:
        if not word[::-1] == word:
            return False
    return True

print(all_palindromes(['a','']))
print(all_palindromes(['ab', 'aaa']))
```

```
True
False
```

In [41]:

```
def process_string(word):
    """
    function to process a given string by either returning the first 3
    or a list of characters depending on the parity of its length.
    Args:
        word (string): a string of characters.
    returns:
        (up to) a 3-character string if length is odd else a list of ch
    >>> process_string('abcde')
    'abc'
    >>> process_string('nicole')
    ['n', 'i', 'c', 'o', 'l', 'e']
    """
    if len(word)%2==1:
        return word[:3]
    else:
        output = []
        for c in word:
            output.append(c)
        return output

print(process_string('abcde'))
print(process_string('nicole'))
```

```
abc
['n', 'i', 'c', 'o', 'l', 'e']
```

In [37]:

```python
# alternative
def process_string(word):
    """
    function to process a given string by either returning the first 3
    or a list of characters depending on the parity of its length.
    Args:
        word (string): a string of characters.
    returns:
        (up to) a 3-character string if length is odd else a list of ch
    >>> process_string('abcde')
    'abc'
    >>> process_string('nicole')
    ['n', 'i', 'c', 'o', 'l', 'e']
    """
    if len(word)%2==1:
        return word[:3]
    else:
        return list(word)

print(process_string('abcde'))
print(process_string('nicole'))
```

```
abc
['n', 'i', 'c', 'o', 'l', 'e']
```

In [42]:

```python
# lucett-plank
def find_ordinal_winner(votes, names):
    """
    Function that takes in a nested list, where each sublist is the sam
    as names. Each sublist represents a ballot ranking participants fro
    Return the winner and loser of the votes, that is the name of the p
    who was rated the highest and the name of the person who was rated

    >>> votes = [
    [2,1,3,4],
    [3,2,1,4],
    [2,1,4,3]
    ]
    >>> names = ['tayven','gwen','linh','brandon']
    >>> winner, loser = find_ordinal_winner(votes,names)
    >>> winner
    'gwen'
    >>> loser
    'brandon'
    """
    scores = [0]*len(names)
    for vote in votes:
        for i in range(len(names)):
            scores[i] += vote[i]
    index_winner = scores.index(min(scores))
    index_loser = scores.index(max(scores))
    return names[index_winner], names[index_loser]

votes = [[2,1,3,4],[3,2,1,4],[2,1,4,3]]
names = ['tayven','gwen','linh','brandon']
```

```
winner, loser = find_ordinal_winner(votes,names)
print(winner)
print(loser)
```

```
gwen
brandon
```