3. Recursivity

```
pros:
        - good for tree traversal, performs better in such tasks
        - reduces time complexity
        - more elegant code
cons:
        - stack unfriendly -> uses more memory
        - usually slower due to overhead of mainting stack
```

]:
```python
def even(x):
    eeven = True
    while(x is not 0):
        eeven, x = not eeven, x-1
    return eeven
```

]:
```python
[even(i) for i in range(0,10)]
```

]:
```
[True, False, True, False, True, False, True, False, True, False]
```

5. Parenthesis Matching

Pseudo Code

```
def parenthesisMatch(string, len):
    - create dict_open that contains opening parenthesis [dicts in python are indexed by keys -> maps hashable
values']
    - create another dict_close that contains closing ones
    - create stack
    - iterate through string, while looking for parenthesis only:
        - if parenthesis: check if its an opening one by comparing it with dict_open items
            - if no: check whether stack is empty:
                    - if yes: return false
                    - if no: check whether it fits to the current one by comparing key of current item in
                            dict_closed with dict_open
            - if yes: stack it.
    - return true if string is completly iterated through.
```