# Assignment 10

#### 1. Create a function that takes the width, height and character and returns a picture frame as a 2D list.

**Examples:**  
**get\_frame(4, 5, "#") ➞ [ ["####"], ["# #"], ["# #"], ["# #"], ["####"] ]**  
**# Frame is 4 characters wide and 5 characters tall.**

get\_frame(10, 3, "\*") ➞ [ ["\*\*\*\*\*\*\*\*\*\*"], ["\* \*"], ["\*\*\*\*\*\*\*\*\*\*"] ]  
# Frame is 10 characters and wide and 3 characters tall.

get\_frame(2, 5, "0") ➞ "invalid"  
# Frame's width is not more than 2.

In [1]:

**def** get\_frame(in\_width,in\_height,in\_character):  
 **if** in\_width **<=** 2:  
 print("Invalid")  
 **else**:  
 out\_list **=** []  
 **for** height **in** range(in\_height):  
 **if** height **==** 0 **or** height **==** in\_height**-**1:  
 out\_list**.**append([in\_width**\***in\_character])  
 **else**:  
 out\_list**.**append([in\_character**+**' '**\***(in\_width**-**2)**+**in\_character])  
 **for** ele **in** out\_list:  
 print(ele)  
 print()  
   
get\_frame(4, 5, "#")  
get\_frame(10, 3, "\*")  
get\_frame(2, 5, "0")

['####']  
['# #']  
['# #']  
['# #']  
['####']  
  
['\*\*\*\*\*\*\*\*\*\*']  
['\* \*']  
['\*\*\*\*\*\*\*\*\*\*']  
  
Invalid

#### 2. Write three functions:

1. boolean\_and
2. boolean\_or
3. boolean\_xor

These functions should evaluate a list of True and False values, starting from the leftmost element and evaluating pairwise. **Examples:**  
**boolean\_and([True, True, False, True]) ➞ False**  
**# [True, True, False, True] => [True, False, True] => [False, True] => False**

boolean\_or([True, True, False, False]) ➞ True  
# [True, True, False, True] => [True, False, False] => [True, False] => True

boolean\_xor([True, True, False, False]) ➞ False  
# [True, True, False, False] => [False, False, False] => [False, False] => False

In [2]:

**def** boolean\_and(in\_list):  
 in\_list\_clone **=** in\_list**.**copy()  
 **while** len(in\_list) **!=** 1:  
 x **=** in\_list**.**pop(0)  
 y **=** in\_list**.**pop(0)  
 in\_list**.**insert(0,(x **and** y))  
 print(f'boolean\_and({in\_list\_clone}) ➞ {in\_list[0]}')  
   
def boolean\_or(in\_list):  
 in\_list\_clone **=** in\_list**.**copy()  
 **while** len(in\_list) **!=** 1:  
 x **=** in\_list**.**pop(0)  
 y **=** in\_list**.**pop(0)  
 in\_list**.**insert(0,(x **or** y))  
 print(f'boolean\_or({in\_list\_clone}) ➞ {in\_list[0]}')  
   
def boolean\_xor(in\_list):  
 in\_list\_clone **=** in\_list**.**copy()  
 **while** len(in\_list) **!=** 1:  
 x **=** in\_list**.**pop(0)  
 y **=** in\_list**.**pop(0)  
 in\_list**.**insert(0,(x **^** y))  
 print(f'boolean\_xor({in\_list\_clone}) ➞ {in\_list[0]}')  
   
boolean\_and([**True**, **True**, **False**, **True**])  
boolean\_or([**True**, **True**, **False**, **False**])  
boolean\_xor([**True**, **True**, **False**, **False**])

boolean\_and([True, True, False, True]) ➞ False  
boolean\_or([True, True, False, False]) ➞ True  
boolean\_xor([True, True, False, False]) ➞ False

#### 3. Create a function that creates a box based on dimension n.

**Examples:**  
**make\_box(5) ➞ [ "#####", "# #", "# #", "# #", "#####" ]**

make\_box(3) ➞ [ "###", "# #", "###" ]

make\_box(2) ➞ [ "##", "##" ]

make\_box(1) ➞ [ "#" ]

In [3]:

**def** make\_box(in\_dimension):  
 out\_list **=** []  
 **for** ele **in** range(in\_dimension):  
 **if** ele **==** 0 **or** ele **==** in\_dimension**-**1:  
 out\_list**.**append('#'**\***in\_dimension)  
 **else**:  
 out\_list**.**append('#'**+**' '**\***(in\_dimension**-**2)**+**'#')  
 **for** ele **in** out\_list:  
 print(ele)  
 print()  
   
make\_box(5)   
make\_box(2)  
make\_box(1)

#####  
# #  
# #  
# #  
#####  
  
##  
##  
  
#

#### 4. Given a common phrase, return False if any individual word in the phrase contains duplicate letters. Return True otherwise.

**Examples:**  
**no\_duplicate\_letters("Fortune favours the bold.") ➞ True**  
**no\_duplicate\_letters("You can lead a horse to water, but you can't make him drink.") ➞ True**  
**no\_duplicate\_letters("Look before you leap.") ➞ False**  
**# Duplicate letters in "Look" and "before".**  
**no\_duplicate\_letters("An apple a day keeps the doctor away.") ➞ False**  
**# Duplicate letters in "apple", "keeps", "doctor", and "away".**

In [4]:

**def** no\_duplicate\_letters(in\_string):  
 out\_put **=** **None**  
 **for** ele **in** in\_string**.**split(' '):  
 **if** len(ele) **==** len(set(ele)):  
 out\_put **=** **True**  
 **else**:  
 out\_put **=** **False**  
 **break**  
 print(f'no\_duplicate\_letters({in\_string}) ➞ {out\_put}')  
   
no\_duplicate\_letters("Fortune favours the bold.")  
no\_duplicate\_letters("You can lead a horse to water, but you can't make him drink.")  
no\_duplicate\_letters("Look before you leap.")  
no\_duplicate\_letters("An apple a day keeps the doctor away.")

no\_duplicate\_letters(Fortune favours the bold.) ➞ True  
no\_duplicate\_letters(You can lead a horse to water, but you can't make him drink.) ➞ True  
no\_duplicate\_letters(Look before you leap.) ➞ False  
no\_duplicate\_letters(An apple a day keeps the doctor away.) ➞ False

#### 5. Write a regular expression that will match the states that voted yes to President Trump's impeachment. You must use RegEx positive lookahead.

**Examples:**  
**txt = "Texas = no, California = yes, Florida = yes, Michigan = no"**  
**pattern = "yourregularexpressionhere"**  
**re.findall(pattern, txt) ➞ ["California", "Florida"]**

In [5]:

**import** re  
txt **=** "Texas = no, California = yes, Florida = yes, Michigan = no"  
pattern **=** r'\w+(?=\s=\syes\*)'  
match **=** re**.**findall(pattern,txt)  
print(f're.findall{pattern,txt} ➞ {match}')

re.findall('\\w+(?=\\s=\\syes\*)', 'Texas = no, California = yes, Florida = yes, Michigan = no') ➞ ['California', 'Florida']