#### Create a function that takes a string and returns a string in which each character is repeated once.

**Examples:**  
double\_char("String") ➞ "SSttrriinngg"  
double\_char("Hello World!") ➞ "HHeelllloo WWoorrlldd!!"  
doublechar("1234!\_") ➞ "11223344!!\_\_"

I/P:

def double\_char(in\_string):

out\_string **=** ''

**for** ele **in** in\_string:

out\_string **+=** ele**\***2

**return** out\_string

print(f'➞ {double\_char("String")}')

print(f'➞ {double\_char("Hello World!")}')

print(f'➞ {double\_char("1234!\_")}')

O/P:

➞ SSttrriinngg

➞ HHeelllloo WWoorrlldd!!

➞ 11223344!!\_\_

#### Create a function that reverses a boolean value and returns the string "boolean expected" if another variable type is given.

**Examples:**  
reverse(True) ➞ False  
reverse(False) ➞ True  
reverse(0) ➞ "boolean expected"  
reverse(None) ➞ "boolean expected"

I/P:

def reverse(in\_bool):

**if** type(in\_bool) **==** bool:

**return** **not** in\_bool

**else**:

**return** "Boolean Expected"

print(f'reverse(True) ➞ {reverse(**True**)}')

print(f'reverse(False) ➞ {reverse(**False**)}')

print(f'reverse(0) ➞ {reverse(0)}')

print(f'reverse(None) ➞ {reverse(**None**)}')

O/P:

reverse(True) ➞ False

reverse(False) ➞ True

reverse(0) ➞ Boolean Expected

reverse(None) ➞ Boolean Expected

#### Create a function that returns the thickness (in meters) of a piece of paper after folding it n number of times. The paper starts off with a thickness of 0.5mm.

**Examples:**  
`num\_layers(1) ➞ "0.001m"

# Paper folded once is 1mm (equal to 0.001m)

num\_layers(4) ➞ "0.008m"

# Paper folded 4 times is 8mm (equal to 0.008m)

num\_layers(21) ➞ "1048.576m"

# Paper folded 21 times is 1048576mm (equal to 1048.576m) `

I/P:

**def** num\_layers(in\_num):

out\_num **=** 0.5

**for** ele **in** range(in\_num):

out\_num **\*=** 2

print(f'Output ➞ {out\_num**/**1000}m')

num\_layers(1)

num\_layers(4)

num\_layers(21)

O/P:

Output ➞ 0.001m

Output ➞ 0.008m

Output ➞ 1048.576m

#### Create a function that takes a single string as argument and returns an ordered list containing the indices of all capital letters in the string.

**Examples:**  
index\_of\_caps("eDaBiT") ➞ [1, 3, 5]  
index\_of\_caps("eQuINoX") ➞ [1, 3, 4, 6]  
index\_of\_caps("determine") ➞ []  
index\_of\_caps("STRIKE") ➞ [0, 1, 2, 3, 4, 5]  
index\_of\_caps("sUn") ➞ [1]

I/P:

**def** index\_of\_caps(in\_string):

out\_string **=** []

**for** ele **in** in\_string:

**if** ele**.**isupper():

out\_string**.**append(in\_string**.**index(ele))

print(f'{in\_string} ➞ {out\_string}')

index\_of\_caps("eDaBiT")

index\_of\_caps("eQuINoX")

index\_of\_caps("determine")

index\_of\_caps("STRIKE")

index\_of\_caps("sUn")

O/P:

eDaBiT ➞ [1, 3, 5]

eQuINoX ➞ [1, 3, 4, 6]

determine ➞ []

STRIKE ➞ [0, 1, 2, 3, 4, 5]

sUn ➞ [1]

#### Using list comprehensions, create a function that finds all even numbers from 1 to the given number.

**Examples:**  
find\_even\_nums(8) ➞ [2, 4, 6, 8]  
find\_even\_nums(4) ➞ [2, 4]  
find\_even\_nums(2) ➞ [2]

I/P:

**def** find\_even\_nums(in\_num):

out\_list **=** [i **for** i **in** range(1,in\_num**+**1) **if** i**%2** == 0]

print(f'Output ➞ {out\_list}')

find\_even\_nums(8)

find\_even\_nums(4)

find\_even\_nums(2)

O/P:

Output ➞ [2, 4, 6, 8]

Output ➞ [2, 4]

Output ➞ [2]