Write a function to return the reverse of the digits of an integer. Raise a ValueError if any other datatype is input. Ignore the sign.

Examples

- reverse_number(10), and reverse_number(10000000), will return 1
- reverse_number(-10) will return -1
- reverse_number(-1234) should return -4321
- reverse_number(1234) should return 4321
- reverse_number(10.0) will `raise ValueError "Invalid input type. Expecting an integer."`

Tasks in the Assignment

- Implement one function: reverse_number`(input)`
- Input: an Integer.
- Output: Sum of the digits.

Code Template (sum_of_digits.py)

```
'''
Reverse a number
'''

def reverse(number):
     pass
```

Test Cases

Function	Inputs	Output	Remarks
reverse	-1.0	ValueError	
reverse	10	1	
reverse	34	43	
reverse	-12345	-54321	
reverse	10000	1	

****

Write a function to return the number of uppercase, lowercase, numerical digits, and other characters (spaces, punctuation marks, emojis, other language characters, etc.) in a string. The input to the function is a string. The return value should be a list of three integers. The return statement looks like this: `return [no_uppercase, no_lowercase, no_numbers, no_others]`

Examples

- string_composition("NITK, Surathkal. Mangalore - 575025.") returns [6, 16, 6, 8]
- string_composition("575025") returns [0, 0, 6, 0]
- string_composition("NITK, Surathkal.") returns [5, 8, 0, 3]
- string_composition("NITK") returns [4, 0, 0, 0]
- string_composition(10.0) will `raise ValueError "Invalid input type. Expecting a String."`

Tasks in the Assignment

- Implement one function: `string_composition(input)`
- Input: a String
- Output: `[no_uppercase, no_lowercase, no_numbers, no_others]`

Code Template (string_composition.py)

```
'''
number of uppercase, lowercase, numerical digits,and other characters in a
string.

'''

def string_composition(input_string):
     pass
```

Test Cases

Function	Inputs	Output	Remarks
string_composition	10.0	ValueError	
string_composition	"NITK"	[4, 0, 0, 0]	
string_composition	"Mangalore - 575025."	[1, 8, 6, 4]	
string_composition	"NITK, Surathkal."	[5, 8, 0, 3]	
string_composition	"575025."	[0, 0, 6, 0]	
string_composition	"NITK, Surathkal. Mangalore - 575025."	[6, 16, 6, 8]	

****

Write a function to return the two roots of a quadratic equation. The input is given in the form of a string "ax2 + bx + c". The return value is a list of one or two real roots. The return statement looks like one of these: `return [root1, root2]` or `return [root1]`. Calculate the roots only if the equation yields non-complex root. Raise a ValueError if the roots are complex. Round off the roots at most at 3 decimal places.

root1 : is $\frac{-b+\sqrt{b**2-4ac}}{2a}$ and root2 is $\frac{-b-\sqrt{b**2-4ac}}{2a}$

- $\Delta = b^2 - 4ac \quad , \quad x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$
- if $(\Delta < 0)$ then roots are complex.
- if $(\Delta = 0)$ then there is only one root.
- if $(\Delta > 0)$ then roots are real.

Examples

- roots("4x2") returns [0]
- roots("4x2 + 4x") returns [0, -1]
- roots("4x2 + 4x + 0.75") returns [-0.25, -0.375]
- string_composition("4x2 + x + 10") will `raise ValueError "Roots are Complex."`
- string_composition([4, 4, 0]) will `raise ValueError "Invalid input type. Expecting a String."`
- If the string is incorrectly formatted, Eg. "4 x2 + 4x" `raise ValueError "Invalid equation"`
- If the string is not a quadratic equation formatted, Eg. "4x + 4" `raise ValueError "Not a Quadratic equation"`

Tasks in the Assignment

- Implement one function: `roots(equation)`
- Input: a String of the form "ax2 + bx + c"
- Output: `[root1] OR [root1, root2]`

Code Template (string_composition.py)

```
'''
Roots of a quadratic equation

'''

def roots(equation):
     pass
```

Test Cases

Function	Inputs	Output

roots	“4x2”	[0]
roots	“4x2 + 4x”	[0, -1]
roots	“4x2 + 4x + 0.75”	[-0.25, -0.375]
roots	“x2 - 1”	[1, -1]
roots	“2x2 - 1“	[0.707, -0.707]
roots	“-4x2 + 4x + 10”	[2.158, -1.158]
roots	“-4x2+4x+10”	[-2.158, 1.158]
roots	“-4x2-4x+0”	[-1, 0]
roots	“4x2 + x + 10”	`ValueError “Roots are Complex.”`
roots	[4, 4, 0]	`ValueError “Invalid input type. Expecting a String.”`
roots	“4 x2 , x + 10”	`ValueError “Invalid equation.”`
roots	“4x + 4”	`ValueError “Not a Quadratic equation.”`

****

CS101. A14 - Sieve of Eratosthenes

Ref: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

Use the Sieve of Eratosthenes to find all the primes from 2 up to a given number.

The Sieve of Eratosthenes is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e. not prime) the multiples of each prime, starting with the multiples of 2. It does not use any division or remainder operation.

Create your range, starting at two and continuing up to and including the given limit. (i.e. [2, limit])

The algorithm consists of repeating the following over and over:

- take the next available unmarked number in your list (it is prime)
- mark all the multiples of that number (they are not prime)

Repeat until you have processed each number in your range.

When the algorithm terminates, all the numbers in the list that have not been marked are prime.

The tests don't check that you've implemented the algorithm, only that you've come up with the correct list of primes. However, do not use division or remainder operations (/, %) and you’ll have mostly implemented the Sieve correctly.

Examples

- sieve(10) will print the following list: [2, 3, 5, 7]
- Input is valid if it is a positive number >=2. Any other number given as input: `raise ValueError "Invalid input."`

Tasks in the Assignment

- Implement one function: `sieve(limit)`
- Input: Integer. This is the max limit of the range [2, limit]
- Output: List of prime numbers from 2 to limit (inclusive)

Code Template (raindrops.py)

```
'''
Sieve of Eratosthenes
'''

def sieve(limit):
     pass
```

Sample output

```
print(sieve(34))
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

print(sieve(100))
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]
```

Test Cases

Function	Inputs	Output	Remarks
convert	-1	ValueError	
convert	10	[2, 3, 5, 7]	
convert	34	[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]	
convert	100	[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]	
convert	10000	[large list - not shown here] check Evaluate.py if you are interested.	

****