Question1

Create a function that takes an integer and returns a list from 1 to the given number, where:

1. If the number **can be divided** evenly by 4, amplify it by 10 (i.e. return 10 times the number).
2. If the number **cannot be divided** evenly by 4, simply return the number.

**Examples**

amplify(4) ➞ [1, 2, 3, 40]

amplify(3) ➞ [1, 2, 3]

amplify(25) ➞ [1, 2, 3, 40, 5, 6, 7, 80, 9, 10, 11, 120, 13, 14, 15, 160, 17, 18, 19, 200, 21, 22, 23, 240, 25]

**Notes**

* The given integer will always be equal to or greater than 1.
* Include the number (see example above).
* To perform this problem with its intended purpose, try doing it with list comprehensions. If that's too difficult, just solve the challenge any way you can.

ANS: You can create the `amplify` function using list comprehension to generate the list with the desired conditions. Here's the implementation:

def amplify(num):

return [n \* 10 if n % 4 == 0 else n for n in range(1, num+1)]

# Test cases

print(amplify(4)) # ➞ [1, 2, 3, 40]

print(amplify(3)) # ➞ [1, 2, 3]

print(amplify(25)) # ➞ [1, 2, 3, 40, 5, 6, 7, 80, 9, 10, 11, 120, 13, 14, 15, 160, 17, 18, 19, 200, 21, 22, 23, 240, 25]

The function `amplify` takes an integer `num` as input. It uses a list comprehension to generate the list from 1 to `num`, where each element is multiplied by 10 if it is divisible evenly by 4 (`n % 4 == 0`), otherwise, the element remains as it is (`else n`). The resulting list is returned as the output.

Question2

Create a function that takes a list of numbers and return the number that's unique.

### Examples

unique([3, 3, 3, 7, 3, 3]) ➞ 7

unique([0, 0, 0.77, 0, 0]) ➞ 0.77

unique([0, 1, 1, 1, 1, 1, 1, 1]) ➞ 0

### Notes

Test cases will always have exactly one unique number while all others are the same.

ANS: from collections import Counter

def unique(numbers):

counter = Counter(numbers)

for num, count in counter.items():

if count == 1:

return num

# Test cases

print(unique([3, 3, 3, 7, 3, 3])) # ➞ 7

print(unique([0, 0, 0.77, 0, 0])) # ➞ 0.77

print(unique([0, 1, 1, 1, 1, 1, 1, 1])) # ➞ 0

Question3

Your task is to create a Circle constructor that creates a circle with a radius provided by an argument. The circles constructed must have two getters getArea() (PIr^2) and *getPerimeter()* (2PI\*r) which give both respective areas and perimeter (circumference).

For help with this class, I have provided you with a Rectangle constructor which you can use as a base example.

### Examples

circy = Circle(11)

circy.getArea()

# Should return 380.132711084365

circy = Circle(4.44)

circy.getPerimeter()

# Should return 27.897342763877365

### Notes

Round results up to the nearest integer.

ANS: import math

class Circle:

def \_\_init\_\_(self, radius):

self.radius = radius

def getArea(self):

return round(math.pi \* self.radius \*\* 2)

def getPerimeter(self):

return round(2 \* math.pi \* self.radius, 2)

# Test cases

circy = Circle(11)

print(circy.getArea()) # ➞ 380

circy = Circle(4.44)

print(circy.getPerimeter()) # ➞ 27.9

Question4

Create a function that takes a list of strings and return a list, sorted from shortest to longest.

### Examples

sort\_by\_length(["Google", "Apple", "Microsoft"])

➞ ["Apple", "Google", "Microsoft"]

sort\_by\_length(["Leonardo", "Michelangelo", "Raphael", "Donatello"])

➞ ["Raphael", "Leonardo", "Donatello", "Michelangelo"]

sort\_by\_length(["Turing", "Einstein", "Jung"])

➞ ["Jung", "Turing", "Einstein"]

### Notes

All test cases contain lists with strings of different lengths, so you won't have to deal with multiple strings of the same length.

ANS: def sort\_by\_length(strings):

return sorted(strings, key=len)

# Test cases

print(sort\_by\_length(["Google", "Apple", "Microsoft"])) # ➞ ["Apple", "Google", "Microsoft"]

print(sort\_by\_length(["Leonardo", "Michelangelo", "Raphael", "Donatello"])) # ➞ ["Raphael", "Leonardo", "Donatello", "Michelangelo"]

print(sort\_by\_length(["Turing", "Einstein", "Jung"])) # ➞ ["Jung", "Turing", "Einstein"]

Question5

Create a function that validates whether three given integers form a **Pythagorean triplet**. The sum of the squares of the two smallest integers must equal the square of the largest number to be validated.

### Examples

is\_triplet(3, 4, 5) ➞ True

# 3² + 4² = 25

# 5² = 25

is\_triplet(13, 5, 12) ➞ True

# 5² + 12² = 169

# 13² = 169

is\_triplet(1, 2, 3) ➞ False

# 1² + 2² = 5

# 3² = 9

### Notes

Numbers may not be given in a sorted order.

ANS: To validate whether three given integers form a Pythagorean triplet, you can first sort the numbers in ascending order. Then, you check if the sum of the squares of the two smallest integers is equal to the square of the largest number. Here's how you can implement the `is\_triplet` function:

def is\_triplet(a, b, c):

# Sort the numbers in ascending order

a, b, c = sorted([a, b, c])

# Check if the sum of squares of two smallest numbers is equal to the square of the largest number

return a\*\*2 + b\*\*2 == c\*\*2

# Test cases

print(is\_triplet(3, 4, 5)) # ➞ True

print(is\_triplet(13, 5, 12)) # ➞ True

print(is\_triplet(1, 2, 3)) # ➞ False

In this implementation, the function `is\_triplet` takes three integers `a`, `b`, and `c` as input. It sorts the numbers in ascending order using `sorted([a, b, c])` and then checks if the sum of squares of the two smallest numbers (a^2 + b^2) is equal to the square of the largest number (c^2). If the condition is met, the function returns `True`, indicating that the three numbers form a Pythagorean triplet; otherwise, it returns `False`.