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.

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

# If the number can be divided evenly by 4, amplify it by 10 (i.e. return 10 times the number).

# 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]

# Declare a list variable, l1

l1 = []

# Get the integer input from the user

num = int(input("Enter a integer"))

# Use for loop to add numbers to the list

for i in range(1,num+1):

if i % 4 == 0:

l1.append(i\*10)

else:

l1.append(i)

# Print the list.

print(l1)

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.

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

# Define a function, function1, which returns a unique value in a list

def function1(l1, s1):

for i in s1:

if l1.count(i) == 1:

return i

# Check with multiple inputs and check the result

list1 = [3, 3, 3, 7, 3, 3] # The output should be 7

# Print the unique number

print(function1(list1, set(list1)))

# Check with multiple inputs and check the result

list1 = [0, 0, 0.77, 0, 0] # The output should be 0.77

# Print the unique number

print(function1(list1, set(list1)))

# Check with multiple inputs and check the result

list1 = [0, 1, 1, 1, 1, 1, 1, 1] # The output should be 0

# Print the unique number

print(function1(list1, set(list1)))

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.

# Should return 380.132711084365

circy = Circle(4.44)

circy.getPerimeter()

# Should return 27.897342763877365

Notes

Round results up to the nearest integer.

# create a Circle constructor that creates a circle with a radius provided by an argument.

# Round results up to the nearest integer.

# two getters getArea() (PIr^2) and getPerimeter() (2PI\*r) which give both areas and perimeter

# Declare a pi variable

pi = 22/7

# Declare a class, Circle

class Circle:

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

self.radius = radius

# Declare a function, getArea() to calculate area of a circle

def getArea(self):

return pi \* self.radius \* self.radius

# Declare a function, getParimeter to calculate parimeter of a circle

def getParimeter(self):

return 2 \* pi \* self.radius

# Create a object of a circle, obj1 and print area of a circle

obj1 = Circle(11)

print("Area of a circle is ",round(obj1.getArea()))

# Create a object of a circle,obj2 and print parimeter of a circle

obj2 = Circle(4.44)

print("Parimeter of a circle is ",round(obj2.getParimeter()))

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.

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

# Define a function, function1 to sort a list of string based on length

def sort\_strings(list1):

list2 = sorted(list1,key=len)

return list2

# Call the function, function1 by calling on various inputs given

# and check the output

l1 = ["Google", "Apple", "Microsoft"]

print(sort\_strings(l1)) # the output should be ["Apple", "Google", "Microsoft"]

l1 = ["Leonardo", "Michelangelo", "Raphael", "Donatello"]

print(sort\_strings(l1)) # the output should be ["Raphael", "Leonardo", "Donatello", "Michelangelo"]

l1 = ["Turing", "Einstein", "Jung"]

print(sort\_strings(l1)) # the output should be ["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.

# 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

# Define a function, function1 which validates Pythagorean triplet

def function1(x,y,z):

if x>y and x>z:

if x\*\*2 == y\*\*2 + z\*\*2:

return True

else:

return False

if y>x and y>z:

if y\*\*2 == x\*\*2 + z\*\*2:

return True

else:

return False

if z>x and z>y:

if z\*\*2 == x\*\*2 + y\*\*2:

return True

else:

return False

# Check with multiple inputs

num1,num2,num3 = 3, 4, 5

print("The numbers form a Pythagorean triplet",function1(num1,num2,num3))

num1,num2,num3 = 13, 5, 12

print("The numbers form a Pythagorean triplet",function1(num1,num2,num3))

num1,num2,num3 = 1, 2, 3

print("The numbers form a Pythagorean triplet",function1(num1,num2,num3))