**Python Programming Basic Assignment-24**

**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.

In [9]:

**from** copy **import** deepcopy

**def** amplify(num):

l**=**[]

**if** (num**>**1):

l**=**[i **for** i **in** range(1,num**+**1)]

l2**=**deepcopy(l)

**for** i **in** l2:

**if** i**%4**==0:

l2**.**insert(l2**.**index(i),i**\***10)

l2**.**remove(i)

print(l2)

In [10]:

amplify(4)

[1, 2, 3, 40]

In [11]:

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]

In [12]:

amplify(3)

[1, 2, 3]

In [13]:

amplify(7)

[1, 2, 3, 40, 5, 6, 7]

**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.

In [14]:

**from** copy **import** deepcopy

**def** unique(l):

l2**=**[]

**for** i **in** l:

l2**.**append(str(i))

l3**=**deepcopy(l2)

**for** i **in** l3:

**if** l3**.**count(i)**==**1:

print(i,end**=**' ')

In [6]:

unique([3, 3, 3, 7, 3, 3])

7

In [7]:

unique([0, 0, 0.77, 0, 0])

0.77

In [8]:

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.

In [20]:

**import** math

**class** Circle:

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

self**.**radius**=**radius

**def** getArea(self):

area**=** (math**.**pi)**\***(self**.**radius)**\***(self**.**radius)

**return** round(area)

**def** getPerimeter(self):

perimeter**=**2**\***math**.**pi**\***self**.**radius

**return** round(perimeter)

In [21]:

circy**=**Circle(11)

print(circy**.**getArea())

380

In [22]:

circy**=**Circle(4.44)

print(circy**.**getPerimeter())

28

**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.

In [43]:

**def** sort\_by\_length(l):

**try**:

new\_list**=**sorted(l,key**=**len)

**except** Exception **as** e:

print(e)

**else**:

**return** new\_list

In [44]:

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

Out[44]:

['Apple', 'Google', 'Microsoft']

In [45]:

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

Out[45]:

['Raphael', 'Leonardo', 'Donatello', 'Michelangelo']

In [46]:

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

Out[46]:

['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.

In [70]:

**def** is\_triplet(**\***args):

l**=**[]

**for** i **in** args:

l**.**append(i)

l2**=**deepcopy(l)

large**=**max(l)

l**.**pop(l**.**index(large))

sum**=**0

**for** i **in** l:

sum**=**sum**+**i**\***i

**if** sum**==**(large**\*\***2):

**return** **True**

**return** **False**

In [71]:

is\_triplet(3, 4, 5)

Out[71]:

True

In [72]:

is\_triplet(13, 5, 12)

Out[72]:

True

In [73]:

is\_triplet(1, 2, 3)

Out[73]:

False