W17-1 Add a column with minor/major

PROBLEM

A table contains a list of **first name** and **last names** and **ages** of different people:

First	Last	Age
Ronan	Ogor	22
Jonathan	Faucher	17
Sievny	Nav	08
Seiha	Hi	86

We represent it in Python as follow:

```
[
["ronan", "Ogor", 22],
["Jonathan", "Faucher", 17],
["Sievny", "Nav", 8],
["Seiha ", "Hi", 86]
]
```

We want to add a new column, to know if the person is major (>=18) or minor (<18)

First	Last	Age	Status
Ronan	Ogor	22	major
Jonathan	Faucher	17	minor
Sievny	Nav	08	minor
Seiha	Hi	86	major

So the result will be:

```
[
["ronan", "Ogor", 22, "major"],
["Jonathan", "Faucher", 17, "minor"],
["Sievny", "Nav", 8, " minor "],
["Seiha ", "Hi", 86, " major "]
]
```

INPUT

- Array of person (first name + last name + age)

OUTPUT

- Array of person (first name + last name + age + minor/major)

CORRECTION

```
persons = eval(input())
for person in persons:
    age = person[2]
    if age>18:
        status = "major"
    else:
        status = "minor"

    person.append(status)

print(persons)
```

W17-2 Student results

INPUT

An **array with the students' scores**:

- Each element of the array is a dictionary with 2 properties :
    - Name (name of the student)
    - Score (score of the student)

Example:

```
[{"name": "Narath", "score": 90} , {"name": "Kunthy", "score": 75} , {"name":
"Sreymom", "score": 95}]
```

OUTPUT

A **dictionary** with the:

- minimum score
- maximum score
- average score

Example:

```
{ "minimum": 75 , "maximum": 95, "average": 95 }
```

STEPS TO DO IT

1. Write function to get index of the min score
2. Write function to get index of the max score
3. Write function to get the average score
4. Create a dictionary with the min, max, average , and print it

FUNCTIONS TO CODE :

Function	**getMaxScoreIndex**
Parameters	**scores** (an array of dictionary)
Return value	**Integer** the INDEX of the maximum score
Example	
	`[{"name": "Narath", "score": 90} , {"name": "Kunthy", "score": 75} , {"name": "Sreymom", "score": 95}]`
	Will return 2, because the maximum score (95) is at index 2

Function	**getMinScoreINdex**
Parameters	**scores** (an array of dictionary)
Return value	**Integer** the INDEX of the minimum score
Example	
	`[{"name": "Narath", "score": 90} , {"name": "Kunthy", "score": 75} , {"name": "Sreymom", "score": 95}]`
	Will return 2, because the minimum score (75) is at index 1

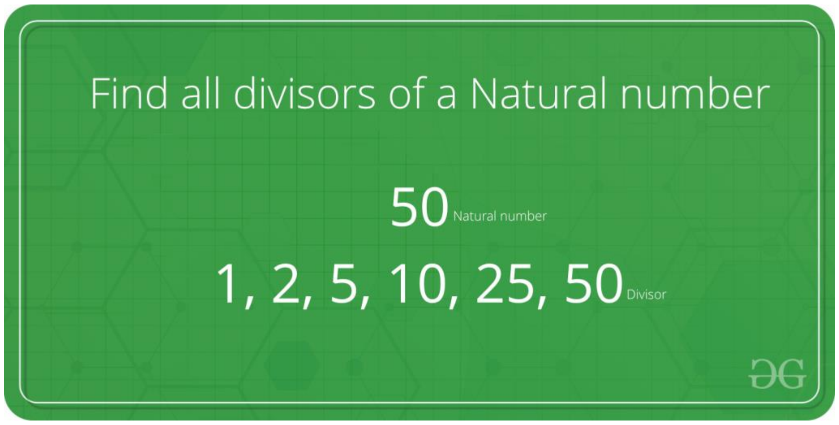
Function	**getAverage**
Parameters	**scores** (an array of dictionary)
Return value	**Integer** the average score (warning: as an integer, not a float!)
Example	
	`[{"name": "Narath", "score": 90} , {"name": "Kunthy", "score": 75} , {"name": "Sreymom", "score": 95}]`
	Will return 86 because the score average is 86

CORRECTION

```
def getMaxScoreIndex(scores) :
    maxScore = 0
    for i in range(len(studentsScores)):
        if(studentsScores[i]["score"] > maxScore):
            maxScore = studentsScores[i]["score"]
    return maxScore

def getMinScoreINdex(scores) :
    minScore = studentsScores[0]["score”]
    for i in range(len(studentsScores)):
        if(studentsScores[i]["score"] < minScore):
            minScore = studentsScores[i]["score"]
    return minScore
```

```
def getAverage(scores) :
    avgScore = 0
    for i in range(len(studentsScores)):
        avgScore += studentsScores[i]["score"]
    return avgScore/len(studentsScores)

# MAIN CODE
studentsScores = eval(input())
maxScore = getMaxScoreIndex(studentsScores)
minScore = getMinScoreINdex(studentsScores)
avgScore = int(getAverage(studentsScores))
print({
    "minimum": minScore,
    "maximum": maxScore,
    "average": avgScore
    })
```

W17-3 Get divisors to divide a numbers to 1


Find all divisors of a Natural number
50 Natural number
1, 2, 5, 10, 25, 50 Divisor

PROBLEM

Given a number as input, print the list of all distinct divisors of it.

EXAMPLES

Input

10

Output

[1, 2 5 10]

Input

100

Output

[1, 2, 4, 5, 10, 20, 25, 50, 100]

FUNCTIONS

You need to implement the following function

Function	canBeDividedBy
Parameters	Integer (the number) Integer (the divisor)
Return value	Boolean - True if the number can be divided by the divisor - False otherwise
Example	canBeDividedBy(12, 3) -> True because 12 can be divided by 3 canBeDividedBy(12, 5) -> False because 12 cannot be divided by 5

CORRECTION

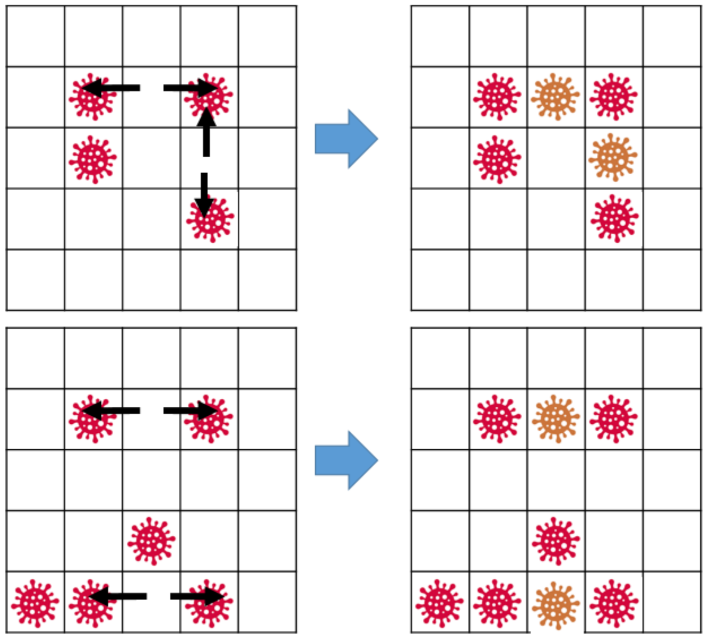
```
def canBeDividedBy(number, divisor) :
    return number % divisor == 0

# MAIN CODE
number = int(input())

result = []
for index in range(1, number+1) :
    if canBeDividedBy(number, index) :
        result.append(index)

print(result)
```

W17-4 COVID contamination

PROBLEM

We represent COVID contamination between people using a grid.

- When a cell is between 2 contaminated cell (horizontally or vertically) then this cell become contaminated



To represent the grid in Python we use an array2D with the following values for cells:

- 1 if cell is contaminated
- 0 if cell is NOT contaminated

```
[
[0, 0, 0, 0, 0],
[0, 1, 0, 1, 0],
[0, 1, 0, 0, 0],
[0, 0, 0, 1, 0],
[0, 0, 0, 0, 0],
]
```

We want to know the final grid after the contamination

INPUTS

- An array 2D of integers *(1 and 0)* : the initial grid
  WARNING: the grid size can change!!!

OUTPUT

- The final grid after contamination

Example:

Input:

```
[[1, 0, 1], [0, 0, 0], [0, 1, 0]]
```

Output

```
[[1, 1, 1], [0, 1, 0], [0, 1, 0]]
```

Explanations

Initial grid is:

```
1, 0, 1
0, 0, 0
0, 1, 0
```

The RED zero is between 2 ones, so this cell will be contaminated

```
1, 1, 1
0, 0, 0
0, 1, 0
```

To code this program, you must follow the following steps:

1- Code the function **isInfected**
2- Code the function **willBeInfected**
3- Code the function **getNextInfectedCells**
4- Update the main program : for each new infected cell, set the cell infected (= 1)

Function	**isInfected(grid, r, c)**
Parameters	grid - array 2D of 1 and 0 r - cell row index c - cell column index
Return value	True if the cell is already infected (before contamination)
Example	grid = [[**1**, 0, 1], [0, 0, 0] , [0, 0, 0]] isInfected (grid, 0, 0) -> True *because the green cell is infected*

Function	**willBeInfected (grid, r, c)**
Parameters	grid - array 2D of 1 and 0 r - cell row index c - cell column index
Return value	True if the cell **will be** infected A cell is infected is either the ones on left/right or the ones on top/bottom are infected
Example	grid = [[1, **0**, 1], [0, 0, 0] , [0, 0, 0]] willBeInfected (grid, 0, 1) -> True *because the orange cell is between 2 infected cells (horizontally* grid = [[1, 0, 0], [**0**, 0, 0] , [1, 0, 0]] willBeInfected (grid, 1, 0) -> True *because the orange cell is between 2 infected cells (vertically)*

Function	**getNextInfectedCells (grid)**
Parameters	grid - array 2D of 1 and 0
Return value	Return the list of cell that will be infected after contamination
Example	grid = [[1, **0**, 1], [0, 0, 0] , [1, **0**, 1],] getNextInfectedCells (grid, 0, 1) -> [[0, 1], [2, 1]] *because 2 cells will be infected (the orange at [0, 1] and the green at [2, 1]* [1, **0**, 1] [0, 0, 0] [1, **0**, 1]

```
# Return True if the cell at given position (row, colum) is infected
def isInfected(grid, r, c ) :
    return grid[r][c] == 1

# Return True if the cell at given position (row, colum) will be infected
after contamination
def willBeInfected(grid, r, c) :

    # 1- check if top cell and bottom cell are  infected (vertical
contamination)
    verticalCont =  r > 0 and r < len(grid) -1 and isInfected(grid, r-1, c)
and isInfected(grid, r+1, c)

    # 2- check if left cell and right cell are  infected (horizontal
contamination)
    horizontalCont = c > 0 and c < len(grid[0]) -1 and isInfected(grid, r, c-
1) and isInfected(grid, r, c+1)

    # 3- the cell will be infected if vertical or horizontal contamination
    return verticalCont or horizontalCont

# Return the list of cell that  will be infected after contamination
# Return is an array of cell positions, each position is an array  [row,
column]
def getNextInfectedCells(grid) :
    rowNb = len(grid)
    columnNb = len(grid[0])
    result = []
    for r in range(rowNb) :
        for c in range(columnNb):
            if not isInfected(grid, r, c) and  willBeInfected(grid, r, c):
                result.append([r, c])

    return result


# MAIN CODE
grid = eval(input())

# Step 1 : we get the list of the cell that will be infected
newInfectedCells = getNextInfectedCells(grid)

# Step 2 : we update the grid (cell infected will be set to 1)
for cell in newInfectedCells:
    row = cell[0]
    column = cell[1]
    grid[row][column] = 1

print(grid)
```