1. Create a function based on the input and output. Look at the examples, there is a pattern.

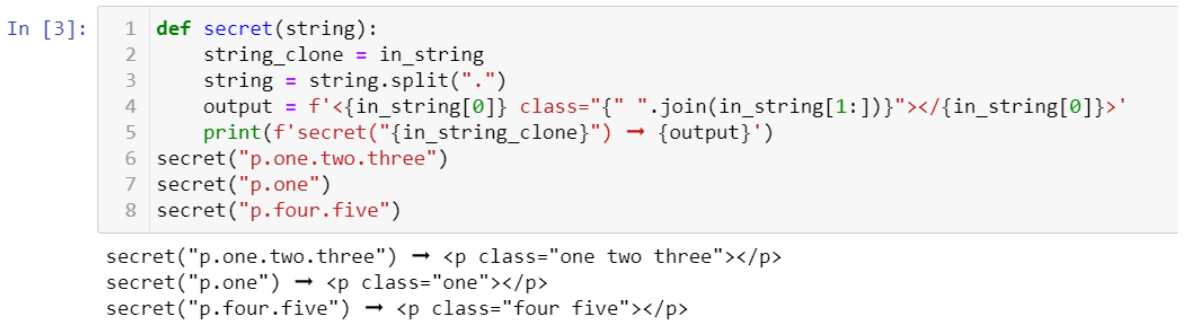
Examples:

secret("p.one.two.three") ➞ "<p class='one two three'></p>"

secret("p.one") ➞ "<p class='one'></p>"

secret("p.four.five") ➞ "<p class='four five'></p>"

Ans:



```
In [3]:
def secret(string):
    string_clone = in_string
    string = string.split(".")
    output = f'<{in_string[0]} class="{" ".join(in_string[1:])}"></{in_string[0]}>'
    print(f'secret("{in_string_clone}") ➞ {output}')
secret("p.one.two.three")
secret("p.one")
secret("p.four.five")
```

```
secret("p.one.two.three") ➞ <p class="one two three"></p>
secret("p.one") ➞ <p class="one"></p>
secret("p.four.five") ➞ <p class="four five"></p>
```

2. Create a function which counts how many lone 1s appear in a given number. Lone means the number doesn't appear twice or more in a row.

Examples:

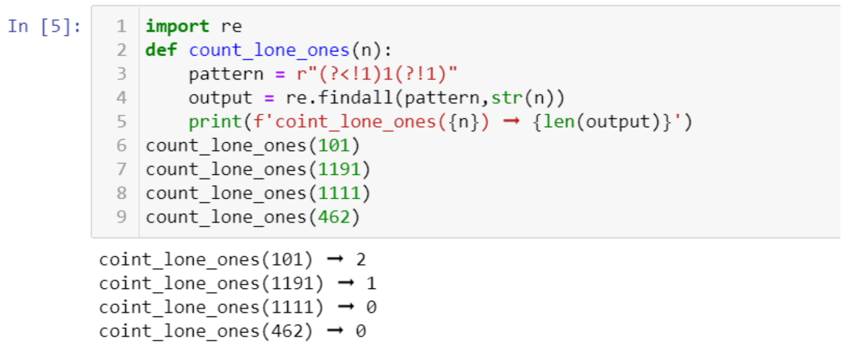
count_lone_ones(101) ➞ 2

count_lone_ones(1191) ➞ 1

count_lone_ones(1111) ➞ 0

count_lone_ones(462) ➞ 0

Ans:



```
In [5]:
import re
def count_lone_ones(n):
    pattern = r"(?<!1)1(?!1)"
    output = re.findall(pattern,str(n))
    print(f'coint_lone_ones({n}) ➞ {len(output)}')
count_lone_ones(101)
count_lone_ones(1191)
count_lone_ones(1111)
count_lone_ones(462)
```

```
coint_lone_ones(101) ➞ 2
coint_lone_ones(1191) ➞ 1
coint_lone_ones(1111) ➞ 0
coint_lone_ones(462) ➞ 0
```

3. Write a method that accepts two integer parameters rows and cols. The output is a 2d array of numbers displayed in column-major order, meaning the numbers shown increase sequentially down each column and wrap to the top of the next column to the right once the bottom of the current column is reached.

Examples:

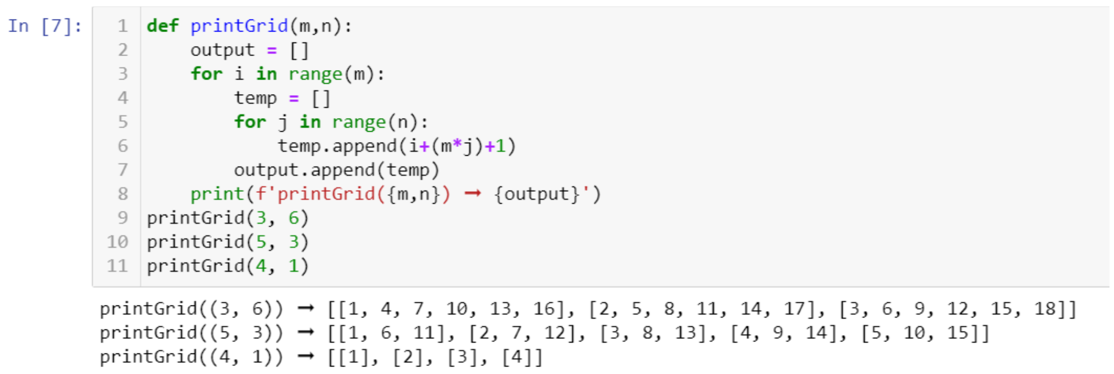
```
printGrid(3, 6) ➞ [
  [1, 4, 7, 10, 13, 16],
  [2, 5, 8, 11, 14, 17],
  [3, 6, 9, 12, 15, 18]
```

]

printGrid(5, 3) ➞ [
  [1, 6, 11],
  [2, 7, 12],
  [3, 8, 13],
  [4, 9, 14],
  [5, 10, 15]
]

printGrid(4, 1) ➞ [
  [1],
  [2],
  [3],
  [4]
]

Ans:



In [7]:

```python
def printGrid(m,n):
    output = []
    for i in range(m):
        temp = []
        for j in range(n):
            temp.append(i+(m*j)+1)
        output.append(temp)
    print(f'printGrid({m,n}) ➞ {output}')
printGrid(3, 6)
printGrid(5, 3)
printGrid(4, 1)
```

```
printGrid((3, 6)) ➞ [[1, 4, 7, 10, 13, 16], [2, 5, 8, 11, 14, 17], [3, 6, 9, 12, 15, 18]]
printGrid((5, 3)) ➞ [[1, 6, 11], [2, 7, 12], [3, 8, 13], [4, 9, 14], [5, 10, 15]]
printGrid((4, 1)) ➞ [[1], [2], [3], [4]]
```

4. Given a list of integers, return the smallest positive integer not present in the list.

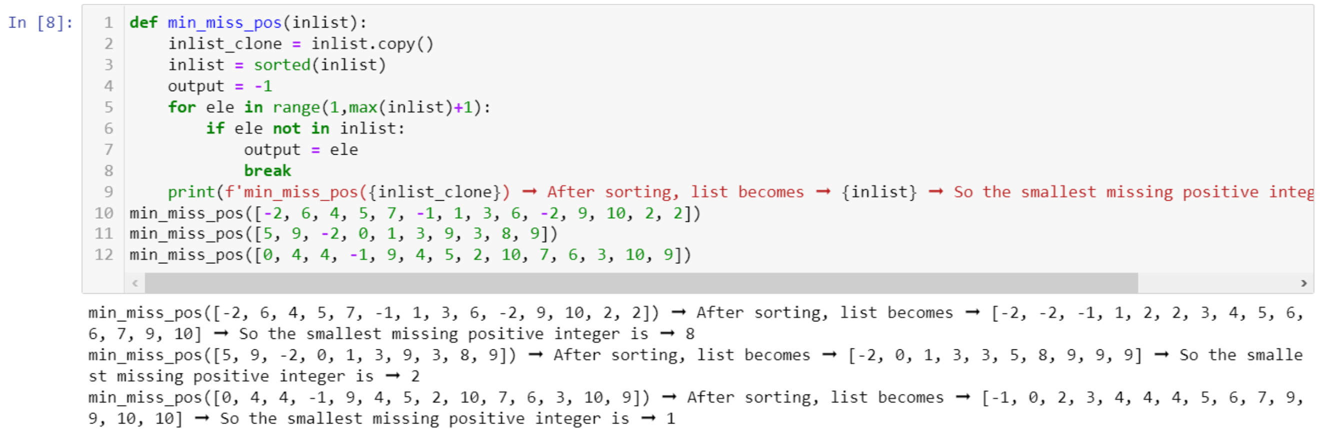
Here is a representative example.
Consider the list: [-2, 6, 4, 5, 7, -1, 7, 1, 3, 6, 6, -2, 9, 10, 2, 2]
After reordering, the list becomes: [-2, -2, -1, 1, 2, 2, 3, 4, 5, 6, 6, 6, 7, 7, 9, 10]
from which we see that the smallest missing positive integer is 8.

Examples:
min_miss_pos([-2, 6, 4, 5, 7, -1, 1, 3, 6, -2, 9, 10, 2, 2]) ➞ 8
# After sorting, list becomes [-2, -2, -1, 1, 2, 2, 3, 4, 5, 6, 6, 7, 9, 10]
# So the smallest missing positive integer is 8

min_miss_pos([5, 9, -2, 0, 1, 3, 9, 3, 8, 9]) ➞ 2
# After sorting, list becomes [-2, 0, 1, 3, 3, 5, 8, 9, 9, 9]
# So the smallest missing positive integer is 2

min_miss_pos([0, 4, 4, -1, 9, 4, 5, 2, 10, 7, 6, 3, 10, 9]) ➞ 1
# After sorting, list becomes [-1, 0, 2, 3, 4, 4, 4, 5, 6, 7, 9, 9, 10, 10]
# So the smallest missing positive integer is 1

Ans:



```
In [8]:
def min_miss_pos(inlist):
    inlist_clone = inlist.copy()
    inlist = sorted(inlist)
    output = -1
    for ele in range(1,max(inlist)+1):
        if ele not in inlist:
            output = ele
            break
    print(f'min_miss_pos({inlist_clone}) ➞ After sorting, list becomes ➞ {inlist} ➞ So the smallest missing positive integ
min_miss_pos([-2, 6, 4, 5, 7, -1, 1, 3, 6, -2, 9, 10, 2, 2])
min_miss_pos([5, 9, -2, 0, 1, 3, 9, 3, 8, 9])
min_miss_pos([0, 4, 4, -1, 9, 4, 5, 2, 10, 7, 6, 3, 10, 9])
```

```
min_miss_pos([-2, 6, 4, 5, 7, -1, 1, 3, 6, -2, 9, 10, 2, 2]) ➞ After sorting, list becomes ➞ [-2, -2, -1, 1, 2, 2, 3, 4, 5, 6,
6, 7, 9, 10] ➞ So the smallest missing positive integer is ➞ 8
min_miss_pos([5, 9, -2, 0, 1, 3, 9, 3, 8, 9]) ➞ After sorting, list becomes ➞ [-2, 0, 1, 3, 3, 5, 8, 9, 9, 9] ➞ So the smalle
st missing positive integer is ➞ 2
min_miss_pos([0, 4, 4, -1, 9, 4, 5, 2, 10, 7, 6, 3, 10, 9]) ➞ After sorting, list becomes ➞ [-1, 0, 2, 3, 4, 4, 4, 5, 6, 7, 9,
9, 10, 10] ➞ So the smallest missing positive integer is ➞ 1
```

5. Google is launching a network of autonomous pizza delivery drones and wants you to create a flexible rewards system (Pizza Points™) that can be tweaked in the future. The rules are simple: if a customer has made at least N orders of at least Y price, they get a FREE pizza!

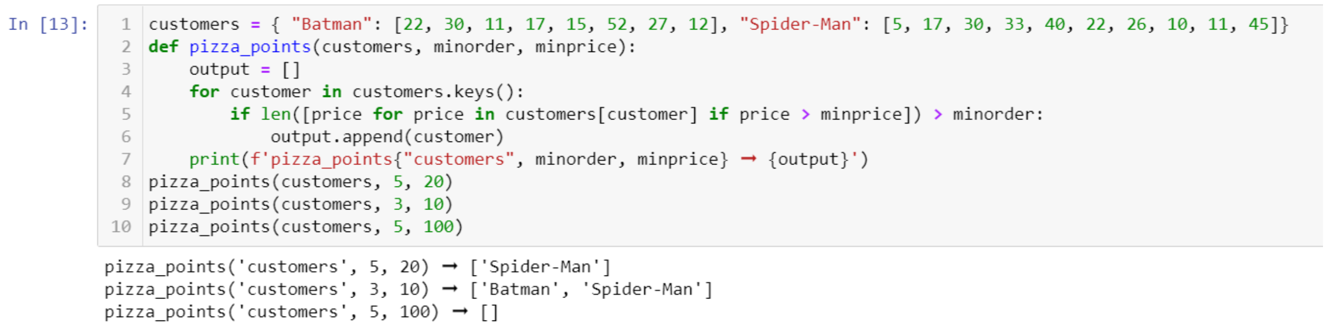
Create a function that takes a dictionary of customers, a minimum number of orders and a minimum order price. Return a list of customers that are eligible for a free pizza.

Examples:

customers = {
  "Batman": [22, 30, 11, 17, 15, 52, 27, 12],
  "Spider-Man": [5, 17, 30, 33, 40, 22, 26, 10, 11, 45]
}

pizza_points(customers, 5, 20) ➞ ["Spider-Man"]
pizza_points(customers, 3, 10) ➞ ["Batman", "Spider-Man"]
pizza_points(customers, 5, 100) ➞ []

Ans:



```
In [13]:
customers = { "Batman": [22, 30, 11, 17, 15, 52, 27, 12], "Spider-Man": [5, 17, 30, 33, 40, 22, 26, 10, 11, 45]}
def pizza_points(customers, minorder, minprice):
    output = []
    for customer in customers.keys():
        if len([price for price in customers[customer] if price > minprice]) > minorder:
            output.append(customer)
    print(f'pizza_points{"customers", minorder, minprice} ➞ {output}')
pizza_points(customers, 5, 20)
pizza_points(customers, 3, 10)
pizza_points(customers, 5, 100)
```

```
pizza_points('customers', 5, 20) ➞ ['Spider-Man']
pizza_points('customers', 3, 10) ➞ ['Batman', 'Spider-Man']
pizza_points('customers', 5, 100) ➞ []
```