Day 6: *Simple User-Defined Functions in Python*

Dot sipped tea in London, checked out several historical and cultural landmarks, and took in the British atmosphere. Then, ready to move on to their next destination, they hurried to the airport to catch their flight. They were running late, and they bit their cheeks nervously as they rushed through security. They had been too distracted, loitering around the old English streets, forgetting that they were on a cross-continental mission! But when they finally got to the gate, they found a large crowd of passengers standing around in a disordered group. The group was grumbling and stamping their feet, wringing their hands, and generally looking unhappy.

Dot pushed through the grumpy crowd to the attendant's desk. "Sorry, our cherished client," the attendant said with a gloomy face, "but the airport is in chaos. The navigators are taking a break from their stations to do a transcendental meditation session, so there's no one available to assign empty gates to incoming planes. Your plane and a dozen others are circling overhead, unable to land." Dot reeled back in shock. Walking over to the window, they saw several empty gates and all the planes anxiously circling overhead. "Worry not, attendant," Dot exclaimed suddenly. "We don't need those navigators. I can help you accomplish this process using data programming!"

Tutorial

In today's challenge you'll use another essential programming concept: **functions**. In the previous challenges and tutorials you've already encountered some functions, like *range()*, *len()*, and *print()*. Those functions are already pre-defined, as in built-in to Python itself. However, pre-defined functions might not always be enough, and sometimes programmers will need to create their own that are designed to suit the task at hand. Today, you'll try creating your own function. Before proceeding, read this article on functions.

```
def name_of_function(<input parameter>, <input parameter 2>):
    '''
    code
    '''
    return output
```

The keyword return doesn't always have to be at the end of the function. We can use it elsewhere, for example if we want to return the first number from a list that's divisible by three.

```
def return_right_number(input_numbers):
    for i in range(len(input_numbers)):
        if input_numbers[i] % 3 == 0:
            return input_numbers[i]
        else :
            continue
    return "Number does not exist"
```

Challenge

The airport uses sensors to keep track of all its airplane gates, and monitor which ones are available. Every time an airplane needs to park at a gate, the system directs them to an available spot for their particular airplane type, or notifies them that no spots are available.

We need to write a function called **find_the_gate()** that returns the coordinates of an available gate, or returns false if there is no available spot. Our function receives a list representing all gates, and a string with the type of the airplane that is looking for a gate.

There are 2 kinds of possible airplanes: **Narrow-Body (Single aisle)** or **Wide-Body**.

- Wide-Body planes can only use gates in **W** spots.
- Narrow-Body planes can use gates in either **N** or **W** spots.

In the list of gates, gates are written in either lower-case orupper-case. An upper-case letter means that the particular gate is **AVAILABLE**, while a lower-case letter means that the gate is **UNAVAILABLE**.

Our function must return a number with the position of the gate where the airplane can park. See the example input and output below for an illustration.

Note: There may be multiple available spots for a particular airplane. If that happens, our function should return the first possible spot to minimize the airplane taxiing time. If there are no available gates, remember to return boolean value False.

Input

```
def find_the_gate(spots, vehicle):
    # Code here!


print(find_the_gate(
  ['w','n','N'], 'narrow'
))

print(find_the_gate(
  ['w','n','N','W','n','W'],'wide'
))

print(find_the_gate(
  ['w','n','n','w','n','n'], 'narrow'
))
```

Expected Output

```
terminal
2
3
False
```

What will be the output of:

```
print(find_the_gate(

['w','n','n','w','N','n','w','N','N','w','n','n','w','n','n','W','W','W','W','n','n
 'wide'
))
```

In [1]:
```
# SOLUTION

def find_the_gate(_list, plane):
    for i in range(len(_list)):
        if plane == "wide":
            if _list[i] == 'W':
                return i
        else:
            if (_list[i] == 'N') or (_list[i] == 'W'):
                return i
    return False
```

In [2]:
```
print(find_the_gate(
  ['w','n','N'], 'narrow'
))

print(find_the_gate(
  ['w','n','N','W','n','W'],'wide'
))

print(find_the_gate(
  ['w','n','n','w','n','n'], 'narrow'
))
```

```
2
3
False
```

In [3]:
```
print(find_the_gate(
  ['w','n','n','w','N','n','w','N','N','w','n','n','w','n','n','W','W','W','W','n','n',
))
```

```
15
```