Python exercises with solutions

June 4, 2020

Table of Contents

1 Reading the row and column of a well location on 24-well plate

A 24-well plate looks like this:

	1	2	3	4	5	6
A						
B						
C						
D						

If we wish to write some code to read the column and row number of any well location, the output would like this:

input = "A4"
output = "A4 is on Row 1 and Column 4"

Let us write some code to enable this readout

1.1 Starting point

[1]:
```python
loc = "A4"
```

1.2 Suggested solution

[2]:
```python
loc = "A4"

row = loc[0]
col = loc[1]

row_map = {
        "A":1,
        "B":2,
        "C":3,
        "D":4
    }

row_no = row_map[row]

output = loc + " is on Row " + str(row_no) + " and Column " + str(col)
print(output)
```

```
A4 is on Row 1 and Column 4
```

2 Swapping input and output in a human readable line

Say out of no reason we would like to change texts of pipette instruction, from:

> Pipette from Well A1 to Well B1
> Pipette from Well A2 to Well B2
> Pipette from Well A3 to Well B3
> Pipette from Well A4 to Well B4
> Pipette from Well A5 to Well B5
> Pipette from Well A6 to Well B6
> Pipette from Well A7 to Well B7
> Pipette from Well A8 to Well B8

to

> Pipette from Well B1 to Well A1
> Pipette from Well B2 to Well A2
> Pipette from Well B3 to Well A3
> Pipette from Well B4 to Well A4

Pipette from Well B5 to Well A5
Pipette from Well B6 to Well A6
Pipette from Well B7 to Well A7
Pipette from Well B8 to Well A8

Let us write a code to automate that process for us

2.1 Starting point

[3]:
```
instructions = [
    "Pipette from Well A1 to Well B1",
    "Pipette from Well A2 to Well B2",
    "Pipette from Well A3 to Well B3",
    "Pipette from Well A4 to Well B4",
    "Pipette from Well A5 to Well B5",
    "Pipette from Well A6 to Well B6",
    "Pipette from Well A7 to Well B7",
    "Pipette from Well A8 to Well B8"
]
```

Before we tackle the entire problem. Let us work on a single line

[4]:
```
oneline = "Pipette from Well A1 to Well B1"
```

Let us try to break down the line to get the information we want, namely, the source and destination info.

[5]:
```
# Code below

broken = oneline.split(" to ")
front = broken[0]
back = broken[1]

# alternatively, the following code would be better
front, back = oneline.split(" to ")

print(front)
print(back)
```

```
Pipette from Well A1
Well B1
```

[6]:
```
# Further split front and back to get the source and dest info

source = front.split("Well ")[1]
print(source)

dest = back.split(" ")[1]
print(dest)
```

```
A1
B1
```

[7]:
```
# Then create a variable to rejoin the source and dest in swapped orders␣
↪together with custom texts
output = "Pipette from Well " + dest + " to Well " + source
output
```

[7]:
```
'Pipette from Well B1 to Well A1'
```

Then, let's apply it to the entire list by using the code together with a for loop

2.2 Suggested solution

[8]:
```
# Code below
for oneline in instructions:
    front, back = oneline.split(" to ")
    source = front.split("Well ")[1]
    dest = back.split(" ")[1]
    output = "Pipette from Well " + dest + " to Well " + source
    print(output)
```

```
Pipette from Well B1 to Well A1
Pipette from Well B2 to Well A2
Pipette from Well B3 to Well A3
Pipette from Well B4 to Well A4
Pipette from Well B5 to Well A5
Pipette from Well B6 to Well A6
Pipette from Well B7 to Well A7
Pipette from Well B8 to Well A8
```

3 Mapping 96-well location to index

In the OT-2 API, well locations within a 96-well plate can be specified as well index in the following way:

	1	2	3	4	5	6	7	8	9	10	11	12
A	0	8	16	24	32	40	48	56	64	72	80	88
B	1	9	17	25	33	41	49	57	65	73	81	89
C	2	10	18	26	34	42	50	58	66	74	82	90
D	3	11	19	27	35	43	51	59	67	75	83	91
E	4	12	20	28	36	44	52	60	68	76	84	92
F	5	13	21	29	37	45	53	61	69	77	85	93
G	6	14	22	30	38	46	54	62	70	78	86	94
H	7	15	23	31	39	47	55	63	71	79	87	95

Let us write a function `location_to_index()` such that a string with conventional well location could be converted into a well index.
For example, `location_to_index("E8")` would give a well index (int) of `60`.

3.1 Starting point

```
[ ]: def location_to_index(loc=str):


         #%%-------Code needed here-----------


         index =
         #%%-------Code end here---------------


         return index

     # Code for checking
     for loc in ["E8", "F4", "H11"]:
         output = "Index of " + loc + " is " + str(location_to_index(loc))
         print(output)
```

3.2 Suggested solution

```
[9]: def location_to_index(loc=str):
         row_char = loc[0]
         col_num = int(loc[1:])

         row_to_num_map = {
             "A":0,
             "B":1,
             "C":2,
             "D":3,
             "E":4,
             "F":5,
             "G":6,
             "H":7
         }

         index = (col_num - 1) * 8 + row_to_num_map[row_char]
         return index

     for loc in ["E8", "F4", "H11"]:
         output = "Index of " + loc + " is " + str(location_to_index(loc))
         print(output)
```

```
Index of E8 is 60
Index of F4 is 29
Index of H11 is 87
```

4 Generating a list of grouped wells

Assuming we want to generate a list of wells of the following 96-well plate, with your desired wells marked in “X”

	1	2	3	4	5	6	7	8	9	10	11	12
A												
B								X	X	X	X	
C								X	X	X	X	
D								X	X	X	X	
E								X	X	X	X	
F								X	X	X	X	
G								X	X	X	X	
H												

Let us code without manually adding all the well names one by one.
We know that the output should look something like:
`["B8","C8","D8","E8", ... ,"E11","F11","G11"]`

4.1 Starting point

[ ]:
```python
# Create list for making component of well locations
rows = []
cols = []

list_of_wells = []

#%%-------Code needed here-----------


#%%-------Code end here-----------

list_of_wells
```

4.2 Suggested solution

[10]:
```python
rows = ["B", "C", "D", "E", "F", "G"]
cols = [8, 9, 10, 11]

list_of_wells = []
for col in cols:
    for row in rows:
        well = row + str(col)
        list_of_wells.append(well)

list_of_wells
```

[10]:
```
['B8',
 'C8',
 'D8',
 'E8',
 'F8',
 'G8',
 'B9',
 'C9',
 'D9',
 'E9',
 'F9',
 'G9',
 'B10',
 'C10',
 'D10',
 'E10',
 'F10',
 'G10',
 'B11',
 'C11',
 'D11',
 'E11',
 'F11',
 'G11']
```

5 Writing a function for calculating factorials

On a scientific calculator we can calculate factorials, for example, $4! = 1 \times 2 \times 3 \times 4 = 24$.

Let's write a function using a for-loop and `range()` to generate factorials for any number we input.

Expected outcome:

input = `factorial(10)`
output = 3628800

5.1 Starting point

[ ]:
```
def factorial(num):

    #%%-------Code needed here-----------
    #%%-------Code end here-----------

    return factorial_of_num

# code for checking
factorial(10)
```

Your output should be 3628800.

5.2 Suggested solution

[11]:
```
def factorial(num):
    factorial_of_num = 1
    for i in range(1, num + 1):
        factorial_of_num = factorial_of_num * i
    return factorial_of_num

# code for checking
factorial(10)
```

[11]:
```
3628800
```