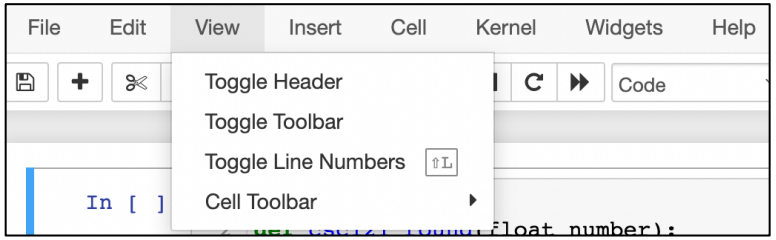
# Lecture 4.2. Functions

**Question 1.** For the code block on the right, write below the print messages in the order in which they appear when the code block is executed.

| Order | Print statement |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |

**Pro tip:** If you ever want to see line numbers for your code cell in Jupyter Notebook, go to *View* in the toolbar and select *Toggle Line Numbers*.




```
1  print('Line 1')
2  def csc121_round(float_number):
3
4      print('Line 4')
5      fractional_part = float_number % 1
6
7      print('Line 7')
8      integer_part = float_number // 1
9
10     print('Line 10')
11     to_ceil = fractional_part >= 0.5
12
13     print('Line 13')
14     rounded = integer_part + to_ceil
15
16     print('Line 16')
17     return rounded
18
19 print('Line 19')
20 number = 2.9
21
22 print('Line 22')
23 number_rounded = csc121_round(number)
24
25 print('Line 25')
26 print(number_rounded)
27
28 print('Line 28')
```

**Question 2.** Starting from line 10 in the code block on the right, fill in the table below tracking:

1. Flow of execution using the first *Line number* column

2. State of variables, for each line number

If the variable is (or becomes) `undefined` at any line of code, state so in the appropriate row of the table below.

```
1  def csc121_round(float_number):
2
3      fractional_part = float_number % 1
4      integer_part = float_number // 1
5      to_ceil = fractional_part >= 0.5
6      rounded = integer_part + to_ceil
7
8      return rounded
9
10 number = 2.9
11 number_rounded = csc121_round(number)
12 print(number_rounded)
```

| Line number | number | float_number | fractional_part | integer_part | to_ceil | rounded | number_rounded |
|---|---|---|---|---|---|---|---|
| 10 | 2.9 | undefined | undefined | undefined | undefined | undefined | undefined |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

**Question 3.** Implement a function `square_root` that computes square root of the input. At the bottom are examples of how the function is expected to be called and the output(s) expected for given input(s).

```
num = 4
num_sqrt = square_root(num)
print(”num_sqrt”) # Should print 2

num = 9
num_sqrt = square_root(num)
print(num_sqrt) # Should print 3

num = 16
num_sqrt = square_root(num)
print(num_sqrt) # Should print 4
```

**Question 4.** Implement a function `euclidean` that accepts four integers `x1, y1, x2, y2` as inputs, representing two points `(x1, y1)` and `(x2, y2)`. **Use `square_root` from Question 3**.

$$euclidean\_distance = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

```
x1 = 1
y1 = 5
x2 = 1
y2 = 5
dist = euclidean(x1, y1, x2, y2)
print(dist) # Should print 0

dist = euclidean(0, 25, 0, 16)
print(dist) # Should print 9
```