1. Python Foundations

The most basic use of Python is as a calculator which takes **expressions** and evaluates their **values**. Python supports all basic arithmetic operations (addition, subtraction, multiplication, division, exponentiation), as well as a few special operations including integer division and the modulo operator. Python will evaluate each line of code, but in a Jupyter notebook only the last expression of each cell is output (shown).

Evaluate the following code cells just as Python would in a Jupyter notebook. If the cell results in an error, write "Error".

a.
```
4 + 2
```

b.
```
8 ** 3
20 // 3
```

c.
```
31 % 2
```

d.
```
5 / 0
51 / 3
```

e.
```
a = 3
b = 4
a = b
a + b
```

f.
```
my_var = 10.0
my_var = my_var + 3
my_var
```

g.
```
magic_number = 3.14
```

h. `"Hello" + "world!"`

i. `"My favorite number is " + 18`

2. Functions

Functions (technically called 'call expressions' in Python) extend the capabilities of Python as a programming language and make managing complex codebases easier. A function takes in zero or more **operands**, evaluates the value of each operand to compute the **arguments**, and performs an operation on those arguments. So before the function can be performed (called), the values of all operands must be computed.

Compute the value of each overall expression. If the expression results in an error, write "Error".

a. `min(-4, 16)`

b. `abs(max(-2, -10))`

c. `min(max(min(max(1, -1), -1), 1), -1)`

d. `abs(3, -7)`

e. `max(100, 100 / 0)`

f. `print("Hello")`

3. Arrays

Arrays are collections of values of the same data type. To create a new array, use the `make_array(...)` function, which can take any number of arguments. All arithmetic operations that can be performed on single values can also be performed with arrays. Array arithmetic is performed **element-wise**.

Compute the value of each expression:

a. `make_array(0, 1, 2)`

b. make$_array()$

c. `len(make_array(3.14, 2.718, 6.28))`

d. `make_array(10, 9, 17) / 2`

e. `make_array(2, 4, 6) + 2`

f. `make_array(5, 7, 9) + make_array(10, 9, 8)`

g. `make_array(3.5, 4.5, 5.5) * make_array(2, 3)`

4. Working with Tables

Tables consist of **rows**, each representing one observation or individual, and **columns**, each of which represents a single attribute and can be represented through an array of values.

`restaurants` table:

Restaurant	Cuisine	Rating	Address
Berkeley Thai House	Thai	4.3	2511 Channing Way
Great China	Chinese	4.4	2190 Bancroft Way
Chez Panisse	French	4.6	1517 Shattuck Ave
Fish & Bird Izakaya	Japanese	4.3	2451 Shattuck Ave

(... 16 rows omitted)

1. Using the table `restaurants`, evaluate the following lines of code:

    a. `restaurants.num_columns`

b. `restaurants.num_rows`

2. Fill in the blanks to return the desired output.

   a. Return all restaurant names in the table `restaurants` as an array

   ```
   restaurants._____(_____)
   ```

   b. Compute the average rating for all restaurants in the `restaurants` table

   ```
   _____(restaurants._____("Rating")
   ```

   c. Find all Thai restaurants in the `restaurants` table

   ```
   restaurants._____("Cuisine",_____)
   ```

   d. Find the lowest rating for all Chinese restaurants in the `restaurants` table

   ```
   np._____(restaurants._____(_____, "Chinese").column(_____))
   ```