Task 9

1. Function

Quiz: Population Density Function

Write a function named `population_density` that takes two arguments, `population` and `land_area`, and returns a population density calculated from those values. I've included two test cases that you can use to verify that your function works correctly. Once you've written your function, use the Test Run button to test your code.

In [2]:
```
def population_density(population , land_area):
    return population / land_area

# test cases for your function
test1 = population_density(10, 1)
expected_result1 = 10
print("expected result: {}, actual result: {}".format(expected_result1, test1))

test2 = population_density(864816, 121.4)
expected_result2 = 7123.6902801
print("expected result: {}, actual result: {}".format(expected_result2, test2))
```

```
expected result: 10, actual result: 10.0
expected result: 7123.6902801, actual result: 7123.690280065897
```

Quiz: `readable_timedelta`

Write a function named `readable_timedelta`. The function should take one argument, an integer `days`, and return a string that says how many weeks and days that is. For example, calling the function and printing the result like this:

```
print(readable_timedelta(10))
```

should output the following:

```
1 week(s) and 3 day(s).
```

In [7]:
```
def readable_timedelta (days):
    x = days //7
    y = days % 7
    return  "{} week(s) and 3 day(s)".format(x ,y)

# test your function
print(readable_timedelta(10))
```

```
1 week(s) and 3 day(s)
```

Quiz: Write a Docstring

Write a docstring for the `readable_timedelta` function you defined earlier! Remember the way you write your docstrings is pretty flexible! Look through Python's docstring conventions **here** and check out this **Stack Overflow page** for some inspiration!

In [8]:
```
def readable_timedelta(days):
    # insert your docstring here
    """ return string of the number of weeks and days included in days.""" # مطلوب

    weeks = days // 7
    remainder = days % 7
    return "{} week(s) and {} day(s)".format(weeks, remainder)
```

Quiz: Lambda with Map

`map()` is a higher-order built-in function that takes a function and iterable as inputs, and returns an iterator that applies the function to each element of the iterable. The code below uses `map()` to find the mean of each list in `numbers` to create the list `averages`. Give it a test run to see what happens.

Rewrite this code to be more concise by replacing the `mean` function with a lambda expression defined within the call to `map()`.

In [10]:
```
numbers = [
                [34, 63, 88, 71, 29],
                [90, 78, 51, 27, 45],
                [63, 37, 85, 46, 22],
                [51, 22, 34, 11, 18]
            ]
averages = list (map(lambda x: sum(x) / len(x) , numbers))
print(averages)
```

```
[57.0, 58.2, 50.6, 27.2]
```

Quiz: Lambda with Filter

`filter()` is a higher-order built-in function that takes a function and iterable as inputs and returns an iterator with the elements from the iterable for which the function returns True. The code below uses `filter()` to get the names in `cities` that are fewer than 10 characters long to create the list `short_cities`. Give it a test run to see what happens.

Rewrite this code to be more concise by replacing the `is_short` function with a lambda expression defined within the call to `filter()`.

```
In [13]: cities = ["New York City", "Los Angeles", "Chicago", "Mountain View", "Denver", "Boston"]
         res = list(filter(lambda y : len(y)<10 ,cities))
         print("short_cities is = ",res)

         short_cities is =  ['Chicago', 'Denver', 'Boston']
```

****

2.Numpy

Quiz: ABC's

Create a numpy array of strings containing letters 'a' through 'j' (inclusive) of the alphabet. Then, use numpy array attributes to print the following information about this array:

1. dtype of array
2. shape of array
3. size of array

The code you submit in the code editor below will not be graded. Use the results from your code below, along with what you remember from the previous video, to complete the quiz below the code editor.

In [16]:
```
import numpy as np

# create numpy array of letters a-j
letter_array = np.array(['a','b','c','d','e','f','g','h','i','j'])
print("Letter Array: ", letter_array)

# get dtype of array
print('dtype',letter_array.dtype)

# get shape of array
print('shape',letter_array.shape)

# get size of array
print('size',letter_array.size)
```

```
Letter Array:  ['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']
dtype <U1
shape (10,)
size 10
```

```
1  import numpy as np
2
3  # Using the Built-in functions you learned about on the
4  # previous page, create a 4 x 4 ndarray that only
5  # contains consecutive even numbers from 2 to 32 (inclusive)
6
7  X =
8
```

In [21]:
```
# Using the Built-in functions you learned about on the
# previous page, create a 4 x 4 ndarray that only
# contains consecutive even numbers from 2 to 32 (inclusive)
import numpy as np

X = np.arange(2,34,2).reshape(4,4)
x
```

Out[21]:
```
array([[ 2.,  4.,  6.,  8.],
       [10., 12., 14., 16.],
       [18., 20., 22., 24.],
       [26., 28., 30., 32.]])
```

```python
import numpy as np

# Use Broadcasting to create a 4 x 4 ndarray that has its first
# column full of 1s, its second column full of 2s, its third
# column full of 3s, etc..

# Do not change the name of this array.
# Please don't print anything from your code! The TEST RUN button below will print your array.
X =

```

In [31]:
```python
# Use Broadcasting to create a 4 x 4 ndarray that has its first
# column full of 1s, its second column full of 2s, its third
# column full of 3s, etc..
# Do not change the name of this array.
# Please don't print anything from your code! The TEST RUN button below will print your array.
import numpy as np
z = np.ones((4,4)) * np.arange(1,5)
print(z)
```

```
[[1. 2. 3. 4.]
 [1. 2. 3. 4.]
 [1. 2. 3. 4.]
 [1. 2. 3. 4.]]
```

****

3.pandas

sun_planets.py | solution.py

```python
import pandas as pd

# DO NOT CHANGE THE VARIABLE NAMES

# Given a list representing a few planets
planets = ['Earth', 'Saturn', 'Venus', 'Mars', 'Jupiter']

# Given another list representing the distance of each of these planets from the Sun
# The distance from the Sun is in units of 10^6 km
distance_from_sun = [149.6, 1433.5, 108.2, 227.9, 778.6]


# TO DO: Create a Pandas Series "dist_planets" using the lists above, representing the distance of the planet from th
# Use the `distance_from_sun` as your data, and `planets` as your index.
dist_planets =


# TO DO: Calculate the time (minutes) it takes light from the Sun to reach each planet.
# You can do this by dividing each planet's distance from the Sun by the speed of light.
# Use the speed of light, c = 18, since light travels 18 x 10^6 km/minute.
time_light =


# TO DO: Use Boolean indexing to select only those planets for which sunlight takes less
# than 40 minutes to reach them.
# We'll check your work by printing out these close planets.
close_planets =
```

In [5]:
```
import pandas as pd

# DO NOT CHANGE THE VARIABLE NAMES

# Given a list representing a few planets
planets = ['Earth', 'Saturn', 'Venus', 'Mars', 'Jupiter']

# Given another list representing the distance of each of these planets from the Sun
# The distance from the Sun is in units of 10^6 km
distance_from_sun = [149.6, 1433.5, 108.2, 227.9, 778.6]


# TO DO: Create a Pandas Series "dist_planets" using the lists above, representing the distance of the planet from the Sun.
# Use the `distance_from_sun` as your data, and `planets` as your index.
dist_planets = pd.Series(data =distance_from_sun , index= planets)
dist_planets

# TO DO: Calculate the time (minutes) it takes light from the Sun to reach each planet.
# You can do this by dividing each planet's distance from the Sun by the speed of light.
# Use the speed of light, c = 18, since light travels 18 x 10^6 km/minute.
time_light = dist_planets / 18
time_light

# TO DO: Use Boolean indexing to select only those planets for which sunlight takes less
# than 40 minutes to reach them.
# We'll check your work by printing out these close planets.
close_planets = time_light[time_light<40]
close_planets
```

Out[5]:
```
Earth     8.311111
Venus     6.011111
Mars     12.661111
dtype: float64
```

Other result to get greater than 40

In [6]:
```
print("grearter than 40 ")
close_planets = time_light[time_light>40]
close_planets
```

```
grearter than 40
```

Out[6]:
```
Saturn     79.638889
Jupiter    43.255556
dtype: float64
```

****