```
import numpy as np
import pandas as pd
import math
```

Q1(1 point): Implement a class for n-sided polygons and a class for points in a Euclidean system, namely *polygon* and *point* respectively. For example, a 4-sided polygon can be defined by 4 points P1, P2, P3, P4, and P1-P4 are each points of the form point(X,Y), and X and Y are coordinates on the X and Y axis, respectively. The edges are listed counterclockwise starting at the lower left: P1 to P2, P2 to P3, P3 to P4, and P4 to P1. The polygon class should work for polygons of any number of edges and have a function perimeter that returns its perimeter (sum of the lengths of the edges).

Hint: use the Pythagorian theorem: if a line segment Z starts at (X1,Y1) and ends at (X2, Y2), the length of Z is the square root of (X1-X2)^2 + (Y1-Y2)^2.

Example: The perimeter of the polygon/triangle on point(1,1), point(1,2), and point(2,2) is 3.4 The perimeter of the 4-sided polygon on point(2,1), point(2,3), point(6,3), and point(4,1) is 10.8

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def __str__(self):
        return '({}, {})'.format(self.x, self.y)

class Polygon:
    def __init__(self, points=[]):
        self.points = points
    def __str__(self):
        asString = "Edges:\n"
        if len(self.points) < 2:
            return '({}, {})'.format(self.points[0].x,
self.points[1].y)
        for i in range(len(self.points)-1):
            asString += "({}, {}) -> ({}, {})\
n".format(self.points[i].x, self.points[i].y, self.points[i+1].x,
self.points[i+1].y)
        asString += "({}, {}) -> ({}, {})\n".format(self.points[-1].x,
self.points[-1].y, self.points[0].x, self.points[0].y)
        return asString
    def perimeter(self):
        per = 0
        if len(self.points) < 2:
            return per
        for i in range(len(self.points)-1):
            x1, x2, y1, y2 = self.points[i].x, self.points[i+1].x,
self.points[i].y, self.points[i+1].y
            per += math.sqrt((x2-x1)**2 + (y2-y1)**2)
        per += math.sqrt((self.points[0].x-self.points[-1].x)**2+
```

```
(self.points[0].y-self.points[-1].y)**2)
        return per


points = [Point(2, 1), Point(2, 3), Point(6, 3), Point(4, 1)]
polygon = Polygon(points)
print(polygon)
print(polygon.perimeter())
```

```
Edges:
(2, 1) -> (2, 3)
(2, 3) -> (6, 3)
(6, 3) -> (4, 1)
(4, 1) -> (2, 1)

10.82842712474619
```

Q2(1 point):

- Create a numpy array X with uniformly random integers from 0 (inclusive) to 100 (exclusive) and shape 1000 * 1000.
- What is the data type for each element in X? Cast the type to int16 and float128 to get two arrays Y and Z, respectively.
- Calculate total bytes consumed by X, Y and Z (using ndarray.nbytes) and explain why.

```
rarr = np.random.uniform(low=0, high=100, size=(1000, 1000))
print("Elements type: {}".format(rarr.dtype))
rarr_int16 = rarr.astype(np.int16)
# np.float128 is platform specific, I am using this on a windows
machine, so I am using np.longdouble
# N.B when np.float128 was used on this machine, it didn't work.
rarr_longdouble = rarr.astype(np.longdouble)
print("float64 array:\t\t{} Bytes".format(rarr.nbytes))
print("int16 array:\t\t{} Bytes".format(rarr_int16.nbytes))
print("longdouble array:\t{} Bytes".format(rarr_longdouble.nbytes))
```

```
Elements type: float64
float64 array:          8000000 Bytes
int16 array:            2000000 Bytes
longdouble array:       8000000 Bytes
```

We have 10^6 elements in our 2D array, if we used the default float64 data type for the elements, we will have 64bits/8= 8 bytes per element, which gives us around 8*10^6 bytes for the overall array.

Same calculation goes for the int16 one, except now each element occupy only 16bit/8=2bytes, which means the whole array occupies around 2*10^6 bytes.

For the last one, which is platform specific, float128 would work on a unix machine, but on my windows x64 machine I couldn't get the float128, however we can go through with the same calculation for whichever datatype we have, #bits/8=#ofbytes, #ofbytes*num_of_elements=total_size_in_bytes

Q3(1 point):

- Write a function call *element_mult* to perform element-wise matrix multiplication.
- Create two random matrices X and Y of size 1000 * 1000
- Multiple X and Y using your element_mult and vectorized multiply (simply using *) and collect the running time (e.g., using time followed by your command). What is ratio of the running time using element-wise and vectorized multiplication?

```
def element_mult(mat1, mat2):
    assert(mat1.shape == mat2.shape)
    mat3 = np.zeros_like(mat1)
    for i in range(mat1.shape[0]):
        for j in range(mat2.shape[1]):
            mat3[i][j] = mat1[i][j]*mat2[i][j]
    return mat3

rmat1 = np.random.rand(1000, 1000)
rmat2 = np.random.rand(1000, 1000)
%timeit rmat3 = element_mult(rmat1, rmat2)
%timeit rmat1 * rmat2
```

```
490 ms ± 25 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
3.11 ms ± 328 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Ratio of running time using element-wise to vectorized multiplication: 448ms/4.53ms = 99

Q4(1 point):

- Import data/AIS/transit_segments.csv, format st_time and end_time in the format: "Thursday February 16 2023 17:30:00"
- For data/AIS/vessel_information.csv, keep only those rows with the type value occurring for at least 100 times in the dataset.
- For data/microbiome_missing.csv, what is the total number of missing data entries? What is the number of rows without any missing data and its ratio with respect to the total number of rows?

```
ts = pd.read_csv('data/AIS/transit_segments.csv')
ts['st_time'] = pd.to_datetime(ts['st_time']).dt.strftime("%A %B %d %Y
%H:%M:%S")
ts['end_time'] = pd.to_datetime(ts['end_time']).dt.strftime("%A %B %d
%Y %H:%M:%S")
ts.head(2)
```

```
   mmsi               name  transit  segment  seg_length  avg_sog
min_sog  \
0     1        Us Govt Ves        1        1         5.1     13.2
```

```
9.2
1     1  Dredge Capt Frank        1        1        13.5     18.6
10.4

   max_sog  pdgt10                             st_time  \
0     14.5    96.5  Tuesday February 10 2009 16:03:00
1     20.6   100.0      Monday April 06 2009 14:31:00

                           end_time
0  Tuesday February 10 2009 16:27:00
1      Monday April 06 2009 15:20:00
```

```
vi = pd.read_csv('data/AIS/vessel_information.csv')
vi_type_counts = vi['type'].value_counts()[vi['type'].value_counts()
>= 100]
vi = vi[vi['type'].isin(vi_type_counts.index)]
vi.head(5)
```

```
   mmsi  num_names              names sov                   flag
flag_type  \
2    21          1      Us Gov Vessel   Y                Unknown
Unknown
3    74          2  Mcfaul/Sarah Bell   N                Unknown
Unknown
5   310          1           Arabella   N               Bermuda
Foreign
6  3011          1         Charleston   N             Anguilla
Foreign
7  4731          1          000004731   N  Yemen (Republic of)
Foreign

   num_loas    loa  max_loa  num_types     type
2         1  208.0    208.0          1  Unknown
3         1  155.0    155.0          1  Unknown
5         1   47.0     47.0          1  Unknown
6         1  160.0    160.0          1    Other
7         1   30.0     30.0          1  Unknown
```

```
mm = pd.read_csv('data/microbiome_missing.csv')
mm.replace(to_replace=['?', 'NA'], value=np.NaN, inplace=True)
mm_filt = mm.dropna()
nan_count = mm.shape[0] - mm_filt.shape[0]
print("Number of rows that contains NaN values: {}".format(nan_count))
print("Number of rows that are full: {}".format((mm.shape[0] -
nan_count)))
print("Ratio of full rows to total: {}".format((mm.shape[0] -
nan_count)/mm.shape[0]))
```

```
Number of rows that contains NaN values: 3
Number of rows that are full: 72
Ratio of full rows to total: 0.96
```

Q5(1 point):

- Merge data/AIS/vessel_information.csv and data/AIS/transit_segments.csv on the "mmsi" column using outer join.
- If you are *not* allowed to call the inner join provided by Pandas but have the above outer join results, how to get the results of inner join? You can use other functions provided by Pandas (but not a function that directly implements the inner join).
- Now directly call the inner join provided by Pandas, check whether your results above are exactly the same.

```
vi = pd.read_csv('data/AIS/vessel_information.csv')
ts = pd.read_csv('data/AIS/transit_segments.csv')
vi_ts_outer = vi.merge(ts, on=['mmsi'], how='outer')
print(vi_ts_outer.shape[0])
```

```
262526
```

```
vi_ts_inner = vi.merge(ts, on=['mmsi'], how='inner')    # Pandas inner
merge
print(vi_ts_inner.shape[0])
```

```
262353
```

```
vi_ts_inner_manual =
vi_ts_outer[(vi_ts_outer['mmsi'].isin(vi['mmsi'])) &
(vi_ts_outer['mmsi'].isin(ts['mmsi']))]   # Manually inner merge
print(vi_ts_inner_manual.shape[0])
```

```
262353
```

Both merges result in a same size dataframe.