Matthew Morris
Professor Kontothanassis
DS 210
26 February 2025

Collaborators: Liam Durcan

Question 1:

```
1. use std::f64::consts::PI;
```

Imports constant pi in floating point 64 format

```
enum Shape {
   Sphere(f64),
   Cube(f64, f64, f64),
   Pyramid(f64, f64, f64),
}
```

Initialises enum shape, with the following details:

The shape *sphere*, with a singular f64 value (radius)

The shapes *cube* and *pyramid*, all with 3 f64 values, (height, side1, side 2)

```
impl Shape {
   fn volume(&self) -> f64 {
       match self {
           Shape::Sphere(radius) => (4.0 / 3.0) * PI * radius.powi(3),
           Shape::Cube(side1, side2, side3) => side1 * side2 * side3,
           Shape::Pyramid(side1,side2, height) => (side1 * side2 * height) / 3.0,
       }
   }
```

Impl creates the implement for the enum, defining the function and its actions

This particular function works for the volume aspect, where it takes a value (&self) -> reading the value given using a reference to this particular spot.

Match self -> based on what the shape is assigned as (sphere, cube, pyramid), takes the respective inputs (side length, or height or radius), and performs the necessary calculations to calculate the volume.

```
  fn surface_area(&self) -> f64 {
       match self {
           Shape::Sphere(radius) => 4.0 * PI * radius.powi(2),
           Shape::Cube(side1, side2, side3) => (2.0 * side1 * side2) + (2.0 * side1 *
side3) + (2.0 * side2 * side3),
           Shape::Pyramid(side1, side2, height) => {
               (side1 * side2) +
               side1 * ((side2/2.0).powi(2) + height.powi(2)).sqrt() +
               side2 * ((side1/2.0).powi(2) + height.powi(2)).sqrt()
           }
       }
   }
```

Same thing as above for the surface area.

```
   fn is_valid(&self) -> bool {
       match self {
           Shape::Sphere(radius) => *radius > 0.0,
           Shape::Cube(side1, side2, side3) => *side1 > 0.0 && *side2 > 0.0 && *side3
> 0.0,
           Shape::Pyramid(side1, side2, height) => *side1 > 0.0 && *side2 > 0.0 &&
*height > 0.0,
       }
   }
```

Does the same thing as above, however acts as a check, and instead of resulting in an f64 data type, it results in a boolean if the shape is > 0 or not.

```
fn create_shape(shape_type: &str, dimensions: Vec<f64>) -> Shape {
   let shape = match shape_type {
       "sphere" => Shape::Sphere(dimensions[0]),
       "cube" => Shape::Cube(dimensions[0], dimensions[1], dimensions[2]),
```

```
        "pyramid" => Shape::Pyramid(dimensions[0], dimensions[1], dimensions[2]),
        _ => panic!("Invalid shape type"),
    };
    if !shape.is_valid() {
        panic!("Invalid shape parameters");
    }
    shape
}
```

Initializes the function create_shape, which allows us to assign string types to the variables “sphere” -> sphere for example. Defines the parameters of which to accept for the main() function, (string, vector). Also implements an else statement (_ => panic!) which ends the code and returns the string “invalid shape type” as a result (string checker if it is not one of the allowed ones). Additionally implements the if function to check if the is valid function is valid (i.e. if the booleans respond as false rather than true), and then it creates the necessary panic if the vector given does not work properly.

```
fn double(shape: &Shape) -> Shape {
    match shape {
        Shape::Cube(side1, side2, side3) => Shape::Cube(side1 * 2.0, side2 * 2.0, side3 * 2.0),
        Shape::Sphere(radius) => Shape::Sphere(radius * 2.0),
        Shape::Pyramid(side1, side2, height) => Shape::Pyramid(side1 * 2.0, side2 * 2.0, height * 2.0),
    }
}
```

The double function pulls the same shape enum (&shape) and matches to the specified shape (pyramid, sphere, etc) and doubles the respective traits (side, radius, etc).

```
fn main() {
    let shape1 = create_shape("sphere", vec![2.0]);
    let shape2 = create_shape("cube", vec![2.0, 2.0, 2.0]);
    let shape3 = create_shape("pyramid", vec![2.0, 2.0, 3.0]);
    let shape4 = double(&shape1);
```

```
    let shape5 = double(&shape2);
    let shape6 = double(&shape3);


    println!("Sphere Volume: {}", shape1.volume());
    println!("Sphere Surface Area: {}", shape1.surface_area());
    println!("Double Sphere Volume: {}", shape4.volume());
    println!("Double Sphere Surface Area: {}", shape4.surface_area());
    println!("Is Sphere Valid: {}", shape1.is_valid());
\\ etc...
}
```

Fn main function assigns each shape (shape1, shape2, etc) where you put the function with the specified string and vector to get volume, surface area, volume. Shape 1-3 are the original 3 vectors, while 4-6 reference shape 1-3 but double their traits.

The print statements are both the shape 1 and shape 1's traits doubled, to find volume and surface area.

Question 1 Results:

Sphere Volume: 33.510321638291124

Sphere Surface Area: 50.26548245743669

Double Sphere Volume: 268.082573106329

Double Sphere Surface Area: 201.06192982974676

Is Sphere Valid: true

Cube Volume: 8

Cube Surface Area: 24

Double Cube Volume: 64

Double Cube Surface Area: 96

Is Cube Valid: true

Pyramid Volume: 4

Pyramid Surface Area: 16.64911064067352

Double Pyramid Volume: 32

Double Pyramid Surface Area: 66.59644256269408

Is Pyramid Valid: true

Question 2:

```
use std::f64::consts::PI;
```

Imports pi from the standard library constants in f64.

```
struct Polygon {
   side: f64,
   side_count: f64,
}


trait Shape {
   fn perimeter(&self) -> f64;
   fn area(&self) -> f64;
   fn radius(&self) -> f64;
   fn apothem(&self) -> f64;
   fn inscribed_circle_area(&self) -> f64;
   fn circumscribed_circle_area(&self) -> f64;
   fn ratio(&self) -> f64;
}
```

Creates struct “polygon,” with side and side_count as float 64 within. Then I created a trait called shape with the respective details area, perimeter, radius, apothem and circle_area, all traits I need to define in order to create the end goal, and what the output of each of the respective traits within shape should give.

```
impl Shape for Polygon {
   fn perimeter(&self) -> f64 {
       self.side * self.side_count
   }
   fn radius(&self) -> f64 {
       self.side / (2.0 * f64::sin(PI / self.side_count))
   }
```

```
    fn apothem(&self) -> f64 {
        self.side / (2.0 * f64::tan(PI / self.side_count))
    }
    fn area(&self) -> f64 {
        (self.side * self.side * self.side_count) / (4.0 * f64::tan(PI /
self.side_count))
    }
    fn inscribed_circle_area(&self) -> f64 {
        let r_inscribed = self.apothem();
        PI * r_inscribed.powi(2)
    }

    fn circumscribed_circle_area(&self) -> f64 {
        let r_circumscribed = self.radius();
        PI * r_circumscribed.powi(2)
    }
    fn ratio(&self) -> f64 {
        self.inscribed_circle_area() / self.area()
    }
}
```

Applies the traits to the shape, while defining exactly what the what the traits should do. (Ex. area for polygon is (side length^2 * side count / (4 tan(pi/# of sides))), or apothem is side length/(2*tan(pi/ #of sides)

```
fn create_shape(side: f64, side_count: f64) -> Box<dyn Shape> {
    Box::new(Polygon{side, side_count})
}
```

Dynamically creates and returns a Polygon within the box <dyn shape>, given the parameters side and sidecount (both f64). Dyn is dynamic dispatch, meaning that it specifies that a trait is being used rather than a type.

Box::new(polygon {side, sidecount}) creates an instance of the polygon struct with the provided values, returning it as a dyn shape.

```
fn main() {
   let mut shapes = Vec::new();
let side_lengths = [1.0, 2.0, 3.0];
let side_counts = [4.0, 8.0, 16.0, 32.0, 64.0, 128.0, 256.0, 512.0, 2048.0, 65536.0];

for &side in &side_lengths {
   for &count in &side_counts {
       shapes.push(create_shape(side, count));
   }
}
for (i, shape) in shapes.iter().enumerate() {
       println!(
           "Shape {} \n Area: {} \n Perimeter: {} \n Radius: {} \n Apothem: {} \n
Inscribed Circle Area: {} \n Circumscribed Circle Area: {} \n Ratio: {}",
           i + 1, shape.area(), shape.perimeter(), shape.radius(), shape.apothem(),
shape.inscribed_circle_area(), shape.circumscribed_circle_area(), shape.ratio());
   }
}
```

Main function assigns the values in the vectors to shapes, iterates through 1.0, 2.0 and 3.0 radii values at each of the side counts.

From assigning each unique shape, it goes on to find the shape's area, perimeter, radius, apothem, inscribed & circumscribed circle area, as well as the ratio from inscribed area to regular area. As shown in the results, it gradually approached 1.

Results from Question 2:

Shape 1

Area: 1.0000000000000002

Perimeter: 4

Radius: 0.7071067811865476

Apothem: 0.5000000000000001

Inscribed Circle Area: 0.7853981633974486
Circumscribed Circle Area: 1.570796326794897
Ratio: 0.7853981633974484
Shape 2
Area: 4.82842712474619
Perimeter: 8
Radius: 1.3065629648763766
Apothem: 1.2071067811865475
Inscribed Circle Area: 4.577635959271528
Circumscribed Circle Area: 5.363034122668977
Ratio: 0.9480594489685199
Shape 3
Area: 20.109357968503396
Perimeter: 16
Radius: 2.5629154477415064
Apothem: 2.5136697460629245
Inscribed Circle Area: 19.850264998127496
Circumscribed Circle Area: 20.635663161524942
Ratio: 0.9871158009727755
Shape 4
Area: 81.2253631008709
Perimeter: 32
Radius: 5.101148618689164
Apothem: 5.076585193804431
Inscribed Circle Area: 80.96423752001814
Circumscribed Circle Area: 81.74963568341558
Ratio: 0.9967851718861698
Shape 5
Area: 325.687481999795
Perimeter: 64
Radius: 10.190008123548056
Apothem: 10.177733812493594
Inscribed Circle Area: 325.4258508897369
Circumscribed Circle Area: 326.2112490531343
Ratio: 0.9991966804850724
Shape 6
Area: 1303.5354839066656

Perimeter: 128
Radius: 20.373878167231435
Apothem: 20.36774193604165
Inscribed Circle Area: 1303.2737265774401
Circumscribed Circle Area: 1304.0591247408377
Ratio: 0.9997991943200188
Shape 7
Area: 5214.927373218955
Perimeter: 256
Radius: 40.7446881033519
Apothem: 40.741620103273085
Inscribed Circle Area: 5214.665584345121
Circumscribed Circle Area: 5215.450982508518
Ratio: 0.9999498000921013
Shape 8
Area: 20860.494900895996
Perimeter: 512
Radius: 81.4878421922257
Apothem: 81.48630820662498
Inscribed Circle Area: 20860.233104136645
Circumscribed Circle Area: 20861.018502300045
Ratio: 0.9999874501175262
Shape 9
Area: 333771.84541562566
Perimeter: 2048
Radius: 325.9494512839691
Apothem: 325.94906778869694
Inscribed Circle Area: 333771.5836164022
Circumscribed Circle Area: 333772.36901456554
Ratio: 0.9999992156341911
Shape 10
Area: 341782637.5264164
Perimeter: 65536
Radius: 10430.378354465194
Apothem: 10430.37834248097
Inscribed Circle Area: 341782637.264617
Circumscribed Circle Area: 341782638.05001515

Ratio: 0.9999999992340178
Shape 11
Area: 4.000000000000001
Perimeter: 8
Radius: 1.4142135623730951
Apothem: 1.0000000000000002
Inscribed Circle Area: 3.1415926535897944
Circumscribed Circle Area: 6.283185307179588
Ratio: 0.7853981633974484
Shape 12
Area: 19.31370849898476
Perimeter: 16
Radius: 2.613125929752753
Apothem: 2.414213562373095
Inscribed Circle Area: 18.31054383708611
Circumscribed Circle Area: 21.452136490675908
Ratio: 0.9480594489685199
Shape 13
Area: 80.43743187401358
Perimeter: 32
Radius: 5.125830895483013
Apothem: 5.027339492125849
Inscribed Circle Area: 79.40105999250999
Circumscribed Circle Area: 82.54265264609977
Ratio: 0.9871158009727755
Shape 14
Area: 324.9014524034836
Perimeter: 64
Radius: 10.202297237378328
Apothem: 10.153170387608862
Inscribed Circle Area: 323.8569500800726
Circumscribed Circle Area: 326.9985427336623
Ratio: 0.9967851718861698
Shape 15
Area: 1302.74992799918
Perimeter: 128
Radius: 20.380016247096112

Apothem: 20.355467624987188
Inscribed Circle Area: 1301.7034035589477
Circumscribed Circle Area: 1304.8449962125371
Ratio: 0.9991966804850724
Shape 16
Area: 5214.141935626662
Perimeter: 256
Radius: 40.74775633446287
Apothem: 40.7354838720833
Inscribed Circle Area: 5213.0949063097605
Circumscribed Circle Area: 5216.236498963351
Ratio: 0.9997991943200188
Shape 17
Area: 20859.70949287582
Perimeter: 512
Radius: 81.4893762067038
Apothem: 81.48324020654617
Inscribed Circle Area: 20858.662337380483
Circumscribed Circle Area: 20861.80393003407
Ratio: 0.9999498000921013
Shape 18
Area: 83441.97960358398
Perimeter: 1024
Radius: 162.9756843844514
Apothem: 162.97261641324997
Inscribed Circle Area: 83440.93241654658
Circumscribed Circle Area: 83444.07400920018
Ratio: 0.9999874501175262
Shape 19
Area: 1335087.3816625027
Perimeter: 4096
Radius: 651.8989025679382
Apothem: 651.8981355773939
Inscribed Circle Area: 1335086.3344656087
Circumscribed Circle Area: 1335089.4760582622
Ratio: 0.9999992156341911
Shape 20

Area: 1367130550.1056657
Perimeter: 131072
Radius: 20860.75670893039
Apothem: 20860.75668496194
Inscribed Circle Area: 1367130549.058468
Circumscribed Circle Area: 1367130552.2000606
Ratio: 0.9999999992340178
Shape 21
Area: 9.000000000000002
Perimeter: 12
Radius: 2.121320343559643
Apothem: 1.5000000000000002
Inscribed Circle Area: 7.068583470577037
Circumscribed Circle Area: 14.137166941154073
Ratio: 0.7853981633974484
Shape 22
Area: 43.4558441227157
Perimeter: 24
Radius: 3.9196888946291293
Apothem: 3.6213203435596424
Inscribed Circle Area: 41.19872363344375
Circumscribed Circle Area: 48.26730710402078
Ratio: 0.94805944896852
Shape 23
Area: 180.98422171653056
Perimeter: 48
Radius: 7.688746343224519
Apothem: 7.5410092381887734
Inscribed Circle Area: 178.65238498314747
Circumscribed Circle Area: 185.72096845372445
Ratio: 0.9871158009727756
Shape 24
Area: 731.028267907838
Perimeter: 96
Radius: 15.30344585606749
Apothem: 15.229755581413292
Inscribed Circle Area: 728.6781376801632

Circumscribed Circle Area: 735.7467211507401
Ratio: 0.9967851718861698
Shape 25
Area: 2931.1873379981553
Perimeter: 192
Radius: 30.57002437064417
Apothem: 30.533201437480784
Inscribed Circle Area: 2928.8326580076323
Circumscribed Circle Area: 2935.9012414782087
Ratio: 0.9991966804850723
Shape 26
Area: 11731.81935515999
Perimeter: 384
Radius: 61.1216345016943
Apothem: 61.103225808124954
Inscribed Circle Area: 11729.463539196962
Circumscribed Circle Area: 11736.532122667539
Ratio: 0.9997991943200188
Shape 27
Area: 46934.34635897059
Perimeter: 768
Radius: 122.2340643100557
Apothem: 122.22486030981925
Inscribed Circle Area: 46931.990259106075
Circumscribed Circle Area: 46939.05884257667
Ratio: 0.999949800092101
Shape 28
Area: 187744.45410806395
Perimeter: 1536
Radius: 244.46352657667708
Apothem: 244.45892461987495
Inscribed Circle Area: 187742.09793722982
Circumscribed Circle Area: 187749.16652070038
Ratio: 0.9999874501175264
Shape 29
Area: 3003946.6087406306
Perimeter: 6144

Radius: 977.8483538519073
Apothem: 977.8472033660908
Inscribed Circle Area: 3003944.2525476194
Circumscribed Circle Area: 3003951.3211310897
Ratio: 0.9999992156341913
Shape 30
Area: 3076043737.737748
Perimeter: 196608
Radius: 31291.135063395584
Apothem: 31291.135027442913
Inscribed Circle Area: 3076043735.381554
Circumscribed Circle Area: 3076043742.4501367
Ratio: 0.999999999234018

What do you observe?

For question 2, as the number of sides increases, the area of the polygon approaches the area of the circumscribed circle. This aligns with n -> infinity, the polygon approximates a perfect circle. Further, the inscribed circle is always smaller, since the inscribed circle always fits inside the polygon. Additionally, at a moderate number of sides (32 - 64), the polygon area is already extremely close to the area of the cirucmscribed circle. By 65536 sides, the difference is negligible. While the absolute values of areas scale with radius squared, the trends follow despite the chosen radius (i.e you get the same observations at side length = 1.0 or 3.0).

Used chatgpt to verify the formulas I used and verify my observations.

https://chatgpt.com/share/67bf81c6-0f64-8006-acbe-e404f8b3ffef