Nathaniel Wolf – CSC 344

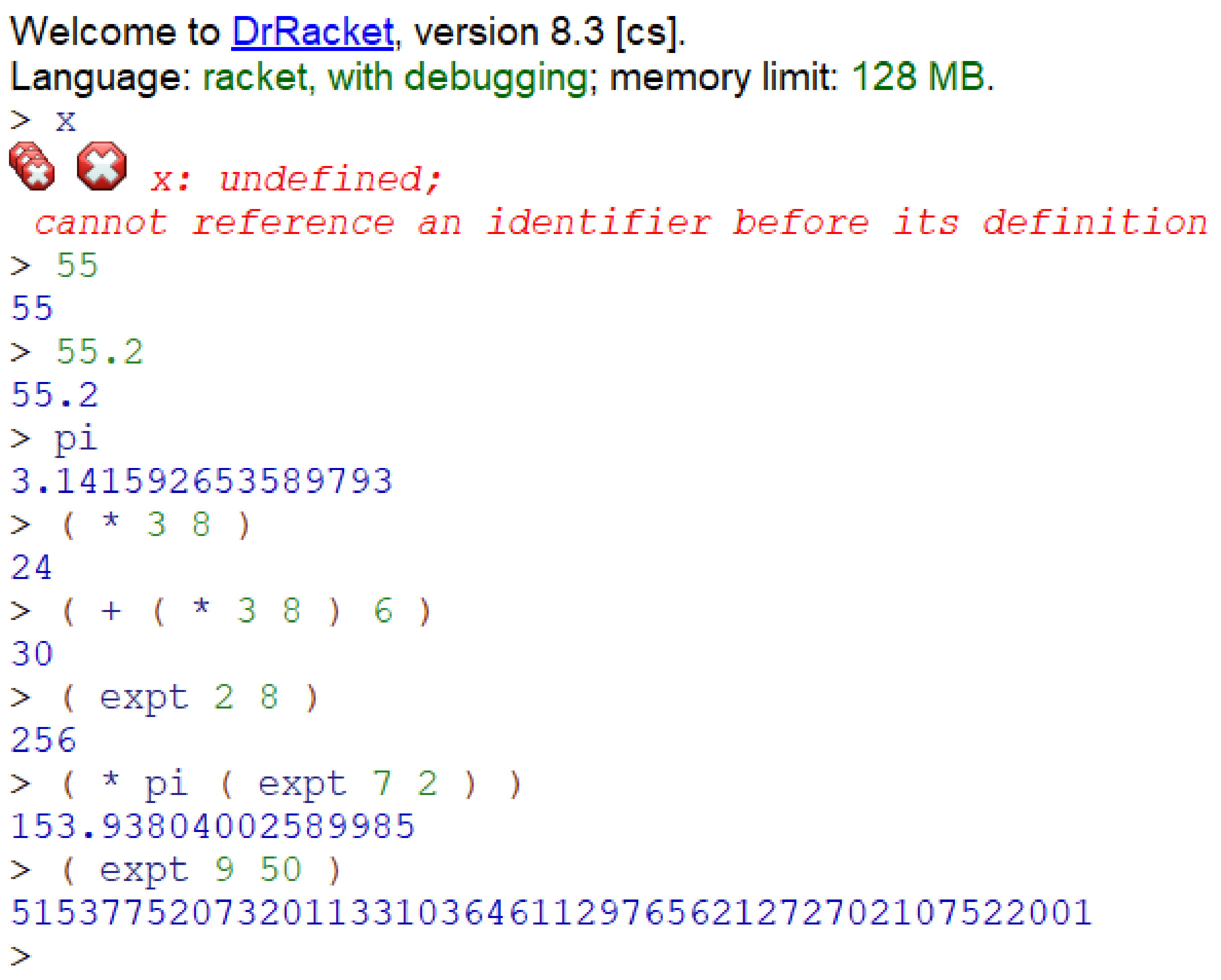
Racket Interactions Assignment 1

Learning Abstract

The purpose of this assignment is to gain an understanding of the basic Racket syntax as well as how to solve basic numerical computation problems. The assignment is conducted exclusively through the DrRacket REPL (that is Read-Evaluate-Print-Loop) with the exception that the language is defined in the upper body. In addition to this I make use of a standard library to make and print shapes such as squares, circles, and rectangles.

Task 1: Mimic Numerical Computation Interactions

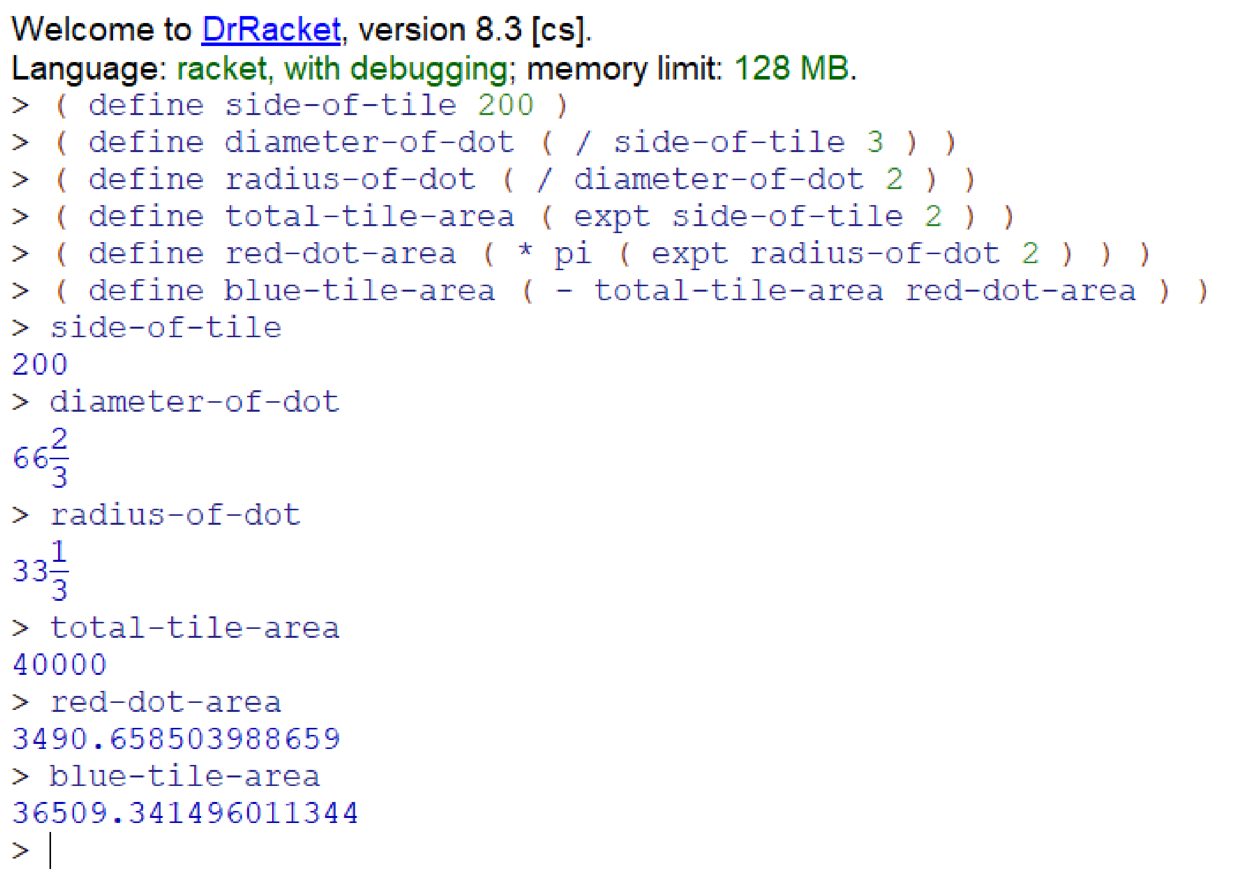
This simply involves mimicking the computations are presented in the assignment documentation. When doing numerical computations racket requires that you evaluate using a prefix notation, that is the operator comes first.



```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> x
x: undefined;
 cannot reference an identifier before its definition
> 55
55
> 55.2
55.2
> pi
3.141592653589793
> ( * 3 8 )
24
> ( + ( * 3 8 ) 6 )
30
> ( expt 2 8 )
256
> ( * pi ( expt 7 2 ) )
153.93804002589985
> ( expt 9 50 )
515377520732011331036461129765621272702107522001
>
```

Task 2: Mimic interactions to solve for area of red and blue dot tile

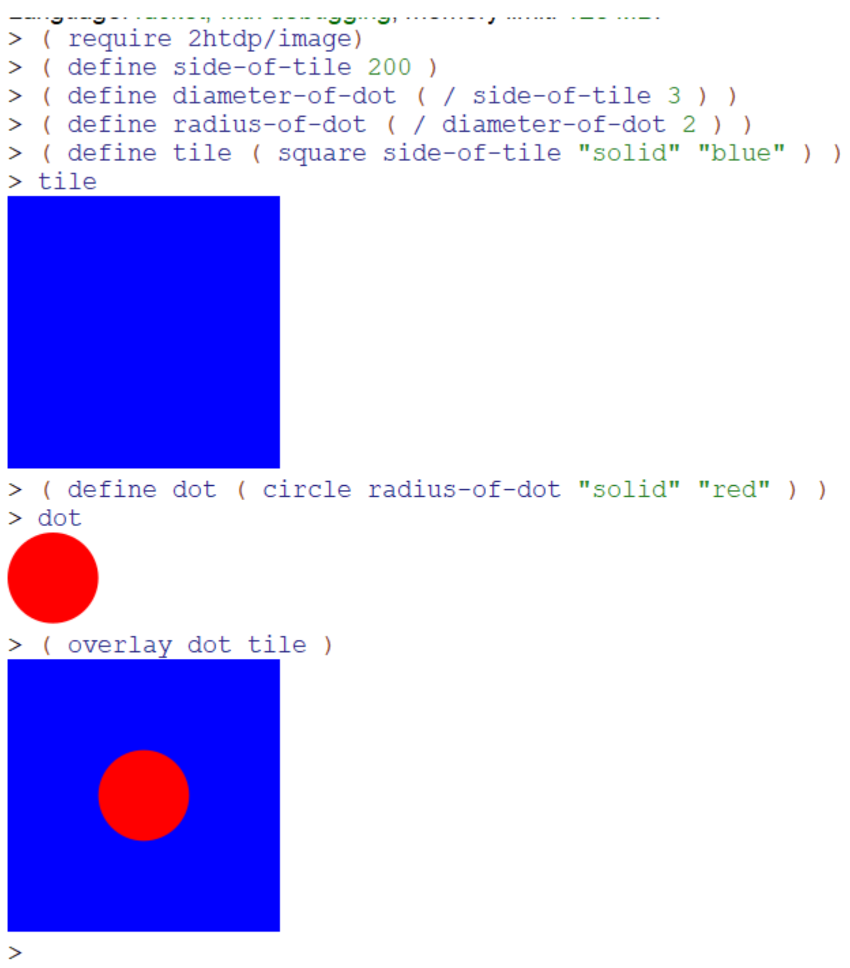
In this part of the assignment, we use the 'define' keyword to initialize our variables while still following the prefix notation convention. By using further defines we can call upon previously stored variables for easier calculation vs longer nested functions. After the definitions are stored and calculated, we call the variable name in the REPL to see its actual value.



```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define side-of-tile 200 )
> ( define diameter-of-dot ( / side-of-tile 3 ) )
> ( define radius-of-dot ( / diameter-of-dot 2 ) )
> ( define total-tile-area ( expt side-of-tile 2 ) )
> ( define red-dot-area ( * pi ( expt radius-of-dot 2 ) ) )
> ( define blue-tile-area ( - total-tile-area red-dot-area ) )
> side-of-tile
200
> diameter-of-dot
66 2/3
> radius-of-dot
33 1/3
> total-tile-area
40000
> red-dot-area
3490.658503988659
> blue-tile-area
36509.341496011344
> 
```

Task 3: Mimic interactions to overlay the dot over the tile

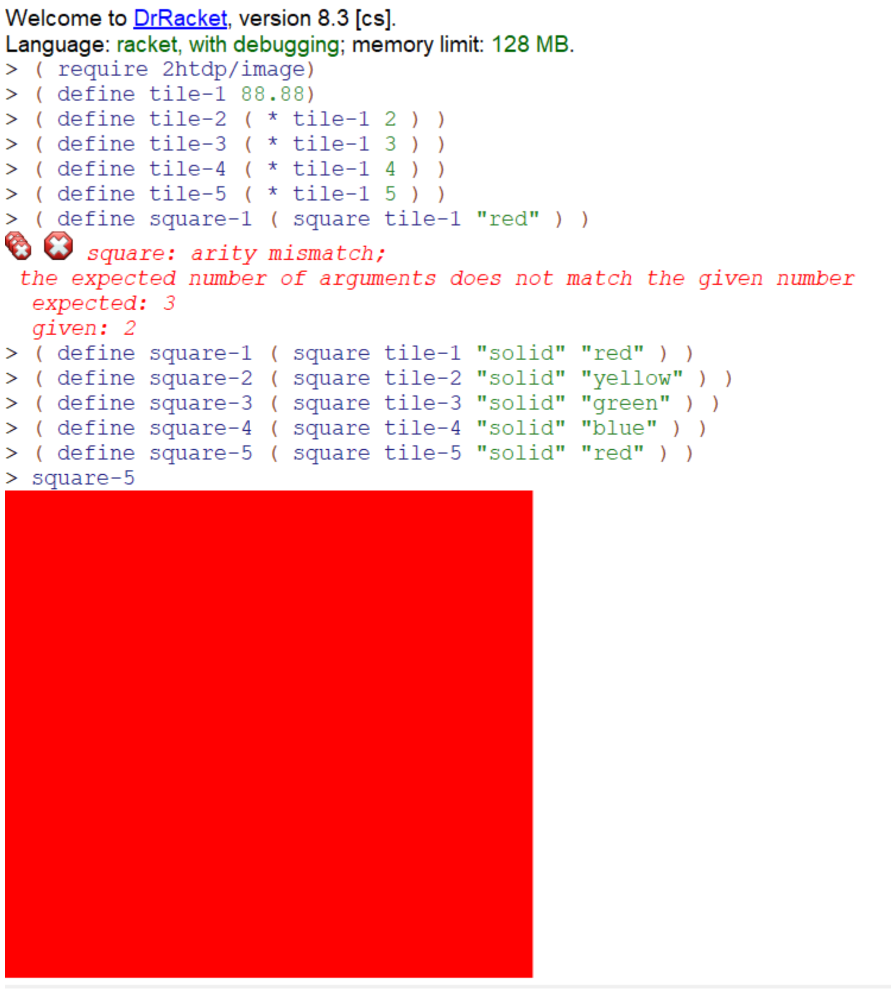
Here we for the first time make use of a DrRacket library which is used to manipulate and display shapes. The variables definitions are conducted the same as in Task2, however the shapes are initialized with specific keywords in this case tile for square, and dot for circle. By calling the overlay function we are able to overlay the dot onto the tile.



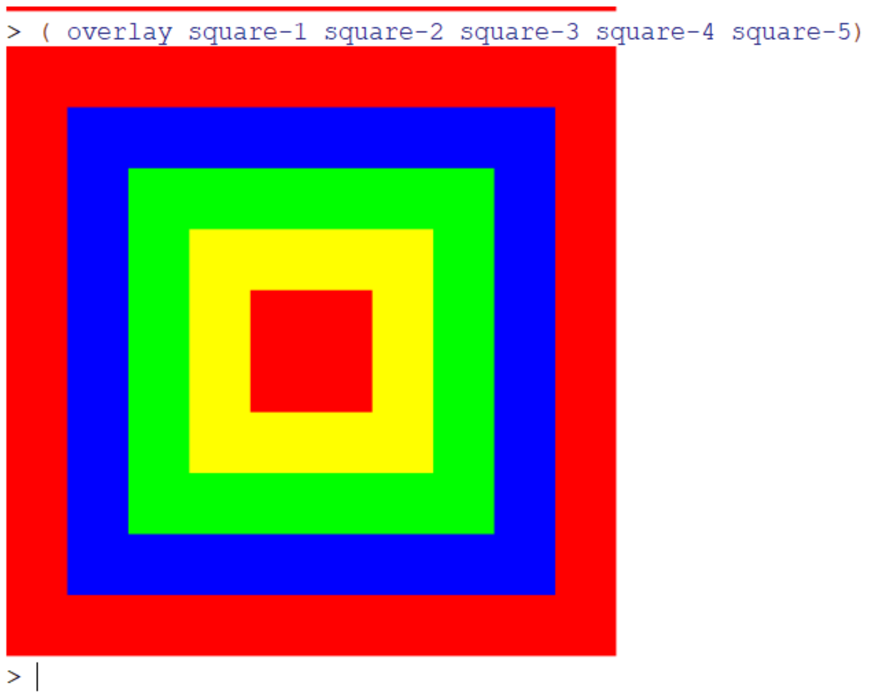
```
> ( require 2htdp/image)
> ( define side-of-tile 200 )
> ( define diameter-of-dot ( / side-of-tile 3 ) )
> ( define radius-of-dot ( / diameter-of-dot 2 ) )
> ( define tile ( square side-of-tile "solid" "blue" ) )
> tile
> ( define dot ( circle radius-of-dot "solid" "red" ) )
> dot
> ( overlay dot tile )
>
```

Task 4: Engage in an interaction session to replicate the concentric square as shown in the assignment documentation.

Here we first define the tile sizes by calling upon an initial value of 88.88 for the first tile's side. Then we multiple each side by and increasing amount of 1 times the initial tile value size until we are at 5 tiles (tile 2: 2 * 88.88, tile 3: 3 * 88.88, etc.) Then using the number definitions we use to define the square objects themselves in which we subsequently call the layer function and layer them squares smallest to greatest to form the concentric squares.



```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( require 2htdp/image)
> ( define tile-1 88.88)
> ( define tile-2 ( * tile-1 2 ) )
> ( define tile-3 ( * tile-1 3 ) )
> ( define tile-4 ( * tile-1 4 ) )
> ( define tile-5 ( * tile-1 5 ) )
> ( define square-1 ( square tile-1 "red" ) )
   square: arity mismatch;
 the expected number of arguments does not match the given number
  expected: 3
  given: 2
> ( define square-1 ( square tile-1 "solid" "red" ) )
> ( define square-2 ( square tile-2 "solid" "yellow" ) )
> ( define square-3 ( square tile-3 "solid" "green" ) )
> ( define square-4 ( square tile-4 "solid" "blue" ) )
> ( define square-5 ( square tile-5 "solid" "red" ) )
> square-5
```

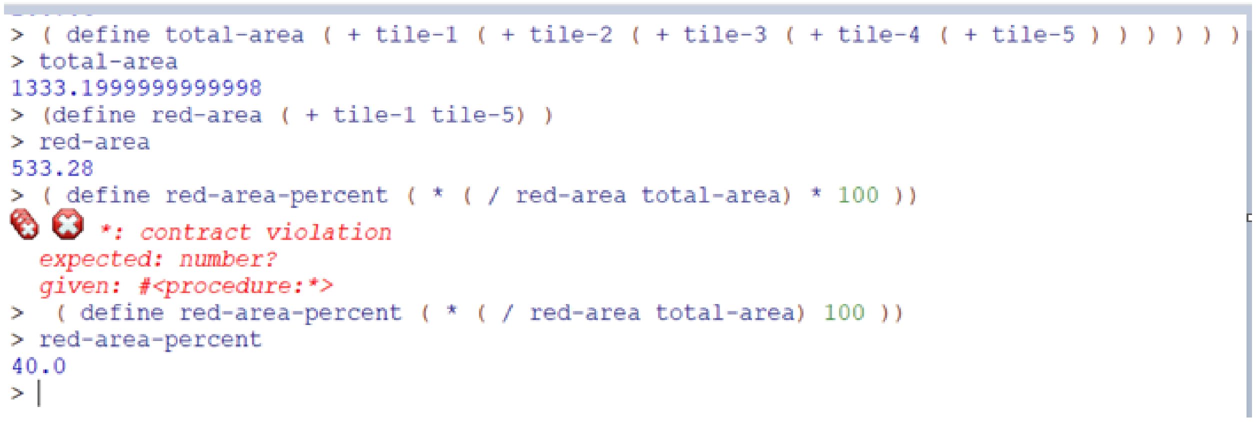


```
> ( overlay square-1 square-2 square-3 square-4 square-5)
```

```
> |
```

Task 5: Engage in an interaction session to evaluate the percentage of the concentric square that contains red

Here we perform mathematical calculations to find the percentage of that the red area occupies. First we define the red area which is outer and inner most part of the square, and then we define the total area. I suppose to reduce some typing I could have simple called the red area variable instead of using tile-1 and tile-5 but it helps to intentionally visualize what I am doing. After that we perform a calculation to get the proportion of red area and convert it to a percent value which ended up being 40%.



```
> ( define total-area ( + tile-1 ( + tile-2 ( + tile-3 ( + tile-4 ( + tile-5 ) ) ) ) ) )
> total-area
1333.1999999999998
> (define red-area ( + tile-1 tile-5) )
> red-area
533.28
> ( define red-area-percent ( * ( / red-area total-area) * 100 ))
    *: contract violation
  expected: number?
  given: #<procedure:*>
>  ( define red-area-percent ( * ( / red-area total-area) 100 ))
> red-area-percent
40.0
> |
```