
Racket Assignment #3: Recursions in Racket

ABSTRACT:

The objective of the third racket assignment was to explore and familiarize oneself with recursion and how to implement it in Racket programming. This assignment includes multiple recursive function codes and demonstrations. By completing this assignment, I have developed a strong foundation in recursive programming, specifically for images and characters. This project serves as an excellent tool to deepen one's knowledge and expertise in recursive functions.

Task 1: Counting Down, Counting Up

Code:

```
#lang racket

( define ( count-down n )
   ( cond
     ( ( < n 0 )
        ( display "" )
     )
     ( ( = n 0 )
        ( display "" )
     )
     ( ( > n 0 )
        ( display n )
        ( display "\n" )
        ( count-down ( - n 1 ) )
     )
   )
)

( define ( count-up n )
   ( cond
     ( ( < n 0 )
        ( display "" )
     )
     ( ( = n 0 )
        ( display "" )
     )
     ( ( > n 0 )
        ( count-up ( - n 1 ) )
        ( display n )
        ( display "\n" )
     )
   )
)
```

Demo:

Language: Determine language from source; memory limit: 128 MB.

```
> ( count-down 5 )

5
4
3
2
1
> ( count-down 10 )
10
9
8
7
6
5
4
3
2
1
> ( count-down 20 )
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
```

```
> ( count-up 5 )
1
2
3
4
5
> ( count-up 10 )
1
2
3
4
5
6
7
8
9
10
> ( count-up 20 )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Task 2: Triangle of Stars

Code:

```
#| TASK 2 |#
( define ( row-of-stars n )
   ( cond
     ( ( = n 0 )
       ( display "\n" )
      )
     ( ( > n 0 )
       ( display "* " )
       ( row-of-stars ( - n 1 ) ) )
     )
   )
)

( define ( triangle-of-stars n )
   ( cond
     ( ( < n 0 )
        ( display "" )
     )
     ( ( = n 0 )
        ( display "" )
     )
     ( ( > n 0 )
        ( triangle-of-stars ( - n 1 ) )
        ( row-of-stars n )
     )
   )
)
```

Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( triangle-of-stars 5 )
*
* *
* * *
* * * *
* * * * *
> ( triangle-of-stars 0 )
> ( triangle-of-stars 15 )
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
>
```

Task 3: Flipping a Coin

Code:

```
#| TASK 3 |#
( define ( flip-coin )
   ( define ht ( random 2 ) )
      ( cond
         ( ( = ht 0 ) 't )
         ( ( = ht 1 ) 'h )
      )
)

( define ( flip-help val n )
   ( cond (
        ( not ( or ( = val n ) ( = val 0 ) ) )
        ( define outcome ( flip-coin ) )
        ( display outcome )
        ( display " " )

      ( cond
         ( ( eq? outcome 'h )
         ( flip-help ( - val 1 ) n )
         )

         ( ( eq? outcome 't)
         ( flip-help ( + val 1 ) n )
         )
        )
      )
   )
)


( define ( flip-for-difference n )
   ( cond
      ( ( < n 0 )
        ( display "" )
      )
      ( ( = n 0 )
        ( display "" )
      )
      ( ( > n 0 )
         ( flip-help n ( * n 2 ) )
      )
   )
)
```

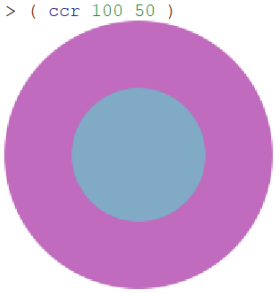
Demo:

```
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
t
> ( flip-for-difference 1 )
t
> ( flip-for-difference 1 )
h
> ( flip-for-difference 2 )
t h h t h t h h
> ( flip-for-difference 2 )
h t h h
> ( flip-for-difference 2 )
t h h h
> ( flip-for-difference 2 )
t t
> ( flip-for-difference 2 )
h h
> ( flip-for-difference 2 )
t h h t t t
> ( flip-for-difference 3 )
t h h h t h h
> ( flip-for-difference 3 )
t h t t t
> ( flip-for-difference 3 )
t t h h h t h h h
> ( flip-for-difference 3 )
t t h h h h t t t t t
> ( flip-for-difference 3 )
h h t h t t t h h h h
> ( flip-for-difference 3 )
t h h h t t h h h
> ( flip-for-difference 4 )
h t h t h h h h
> ( flip-for-difference 4 )
h h t t h t t t h h t t h h h t h t h h t h t h t t t h t h h t h h t h h t h h
> ( flip-for-difference 4 )
h t t t h h h h t h h t h h
> ( flip-for-difference 4 )
h h h h
> ( flip-for-difference 4 )
t h t t h t h t t h h t t h h h t t t h h h t t h h h t t t h t t t h h h t t h t t h t t h t t h h t h h
t t h h t t t t
> ( flip-for-difference 4 )
t h h t h h t h h h
> ( flip-for-difference 4 )
t h t h t h h h h t h t h t h h t t h t h h h
> ( flip-for-difference 4 )
t t t t
>
```

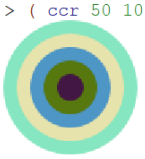
Task 4: Laying Down Colorful Concentric Disks

CCR Demo:

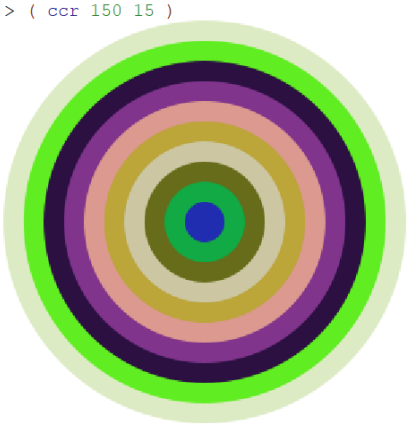
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ccr 100 50 )
```



```
> ( ccr 50 10 )
```



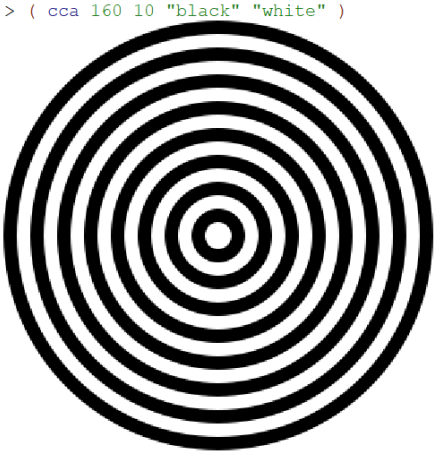
```
> ( ccr 150 15 )
```



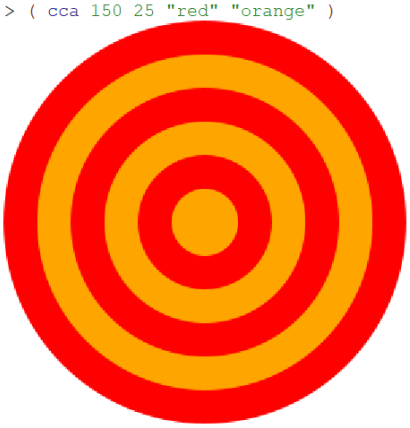
```
>
```

CCA Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( cca 160 10 "black" "white" )
```



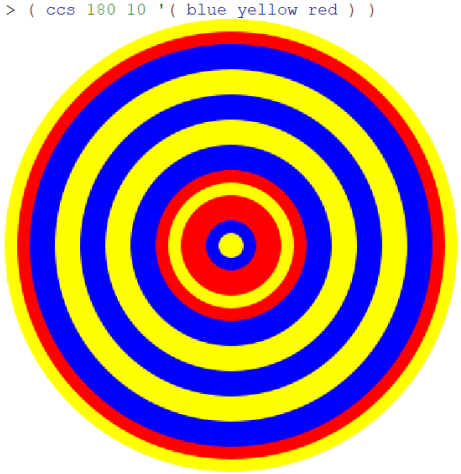
```
> ( cca 150 25 "red" "orange" )
```



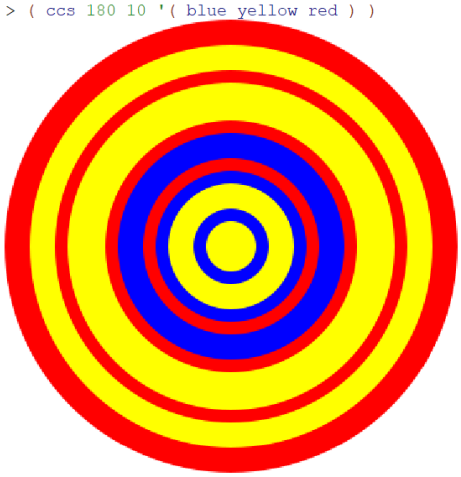
```
>
```

CCS Demo 1:

```
> ( ccs 180 10 '( blue yellow red ) )
```

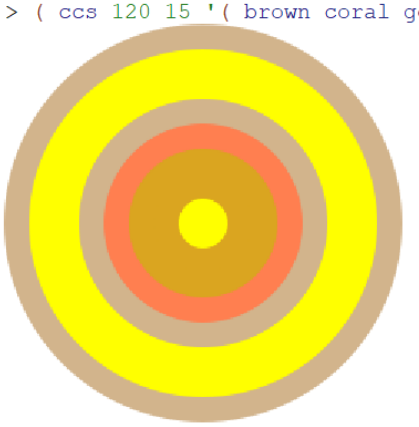


```
> ( ccs 180 10 '( blue yellow red ) )
```



CCS Demo 2:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ccs 120 15 '( brown coral goldenrod yellow olive tan ) )
```



```
> ( ccs 120 15 '( brown coral goldenrod yellow olive tan ) )
```

```
>
```

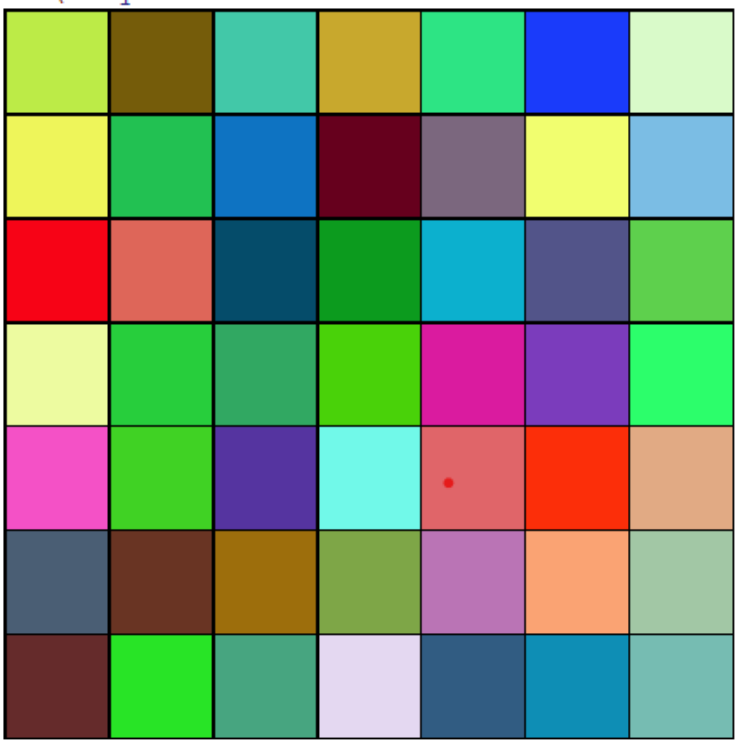
Code:

```
#| CCR |#
( define ( ccr radius diff-rad )
   (cond
     ( ( > radius 0 )
         ( define ( rgb ) ( random 0 256 ) )
         ( define ( random-color ) ( color ( rgb ) ( rgb ) ( rgb ) ) )
         ( overlay ( ccr ( - radius diff-rad ) diff-rad ) ( circle radius "solid" ( random-color ) ) )
      )
      ( ( = radius 0 )
         empty-image )

   )
)

#| CCA |#
( define ( cca radius diff-rad c1 c2 )
   ( cond
      ( ( < radius 0 )
        empty-image )
      ( ( = radius 0 )
        empty-image )
      ( ( > radius 0 )
         ( cca-help radius diff-rad c1 c2 1 )
      )
    )
   )

( define ( cca-help radius diff-rad c1 c2 alt-num )
   (cond
     ( ( = radius 0 )
         empty-image )

     ( ( > radius 0 )
      ( cond
         ( ( = alt-num 1 )
         ( overlay ( cca-help ( - radius diff-rad ) diff-rad c1 c2 2 ) ( circle radius "solid" c1 ) )
         )

         ( ( = alt-num 2 )
         ( overlay ( cca-help ( - radius diff-rad ) diff-rad c1 c2 1 ) ( circle radius "solid" c2 ) )
         )
        )
      )

    )
)

#| CCS |#
( define ( ccs radius diff-rad c )
   ( ccs-help radius diff-rad c ( length c ) )
 )

( define ( ccs-help radius diff-rad c c-num )
   (cond
     ( ( = radius 0 )
         empty-image )

     ( ( > radius 0 )
       ( define (c-ran) ( random c-num ) )
       ( overlay ( ccs-help ( - radius diff-rad ) diff-rad c c-num ) ( circle radius 'solid ( list-ref c ( c-ran ) ) ) )
      )
    )
)
```

Task 5: Variations on Hirst Dots

Random Colored Tile Demo:

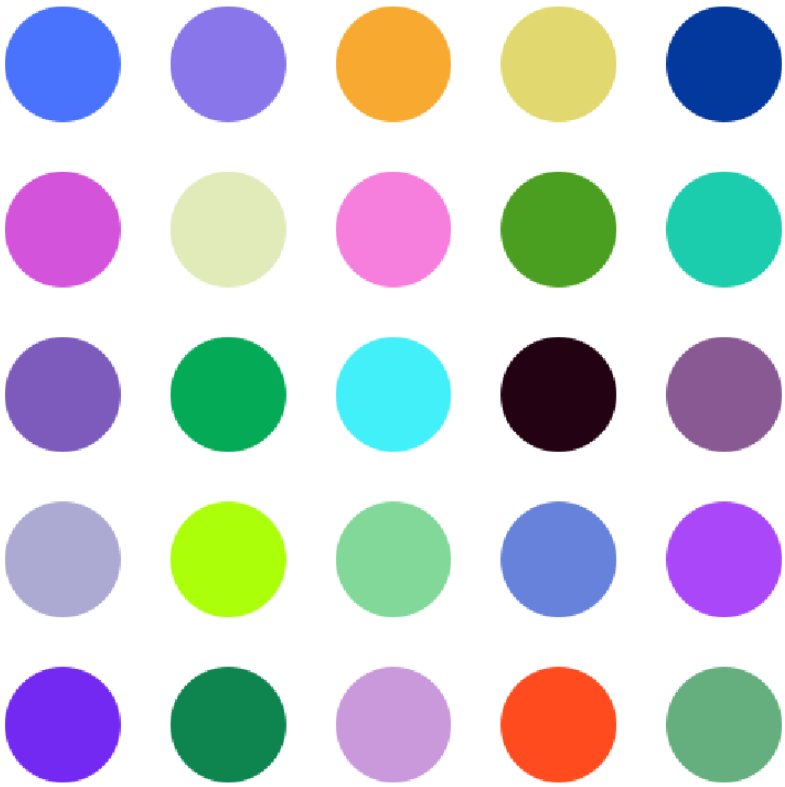
```
Language: racket, with debugging; memory limit: 128 MB.
> ( square-of-tiles 7 random-color-tile )
```



```
>
```

Hirst Dots Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( square-of-tiles 5 dot-tile )
```



```
>
```

CCS Dots Demo:

Language: racket, with debugging; memory limit: 128 MB.
> (square-of-tiles 7 ccs-tile)

>

Nested Diamonds Demo:

Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.

```
> ( square-of-tiles 6 diamond-tile )
```

```
>
```

Unruly Squares Demo:

Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.

```
> ( square-of-tiles 6 wild-square-tile )
```

```
>
```

Code:

```
175
176 #| TASK 5 |#
177 ( define ( row-of-tiles n tile )
178    ( cond
179       ( ( = n 0 )
180        empty-image
181        )
182       ( ( > n 0 )
183         ( beside ( row-of-tiles ( - n 1 ) tile ) ( tile ) )
184         )
185       )
186    )
187
188 ( define ( rectangle-of-tiles r c tile )
189    ( cond
190       ( ( = r 0 )
191         empty-image
192         )
193       ( ( > r 0 )
194         ( above
195           ( rectangle-of-tiles ( - r 1 ) c tile ) ( row-of-tiles c tile ) )
196         )
197       )
198    )
199
200 ( define ( square-of-tiles n tile )
201    ( rectangle-of-tiles n n tile )
202    )
203
204 ( define ( random-color-tile )
205    ( overlay
206      ( square 40 "outline" "black" )
207      ( square 40 "solid" ( random-color ) )
208      )
209    )
210
211 ( define ( random-color )
212    ( define ( rgb ) ( random 0 256 ) )
213    ( color ( rgb ) ( rgb ) ( rgb ) )
214    )
215
216 ( define ( dot-tile )
217    ( overlay
218      ( circle 35 "solid" ( random-color ) )
219      ( square 100 "solid" "white" )
220      )
221    )
```

```
( define ( ccs-tile )
   ( define colors ( random-colors 3 ) )
   ( overlay
     ( ccs 35 5 colors )
     ( square 100 "solid" "white" )
     )
   )

( define ( random-colors n )
   ( cond ( ( > n 0 )
            ( cons ( random-color ) ( random-colors ( - n 1 ) ) )
           )
          ( ( = n 0 ) empty )
         )
   )

( define ( diamond-tile )
   ( define diamondColor ( random-color ) )
   ( overlay ( rotate 45 ( square 30 "solid" "white" ) )
             ( rotate 45 ( square 40 "solid" diamondColor ) )
             ( rotate 45 ( square 50 "solid" "white" ) )
             ( rotate 45 ( square 60 "solid" diamondColor ) )
             ( square 100 "solid" "white" )
            )
   )

( define ( wild-square-tile )
   ( define squareColor ( random-color ) )
   ( define angle ( random 0 90 ) )
   ( overlay
     ( rotate angle ( square 30 "solid" "white" ) )
     ( rotate angle ( square 40 "solid" squareColor ) )
     ( rotate angle ( square 50 "solid" "white" ) )
     ( rotate angle ( square 60 "solid" squareColor ) )
     ( square 100 "solid" "white" )
     )
   )

```