

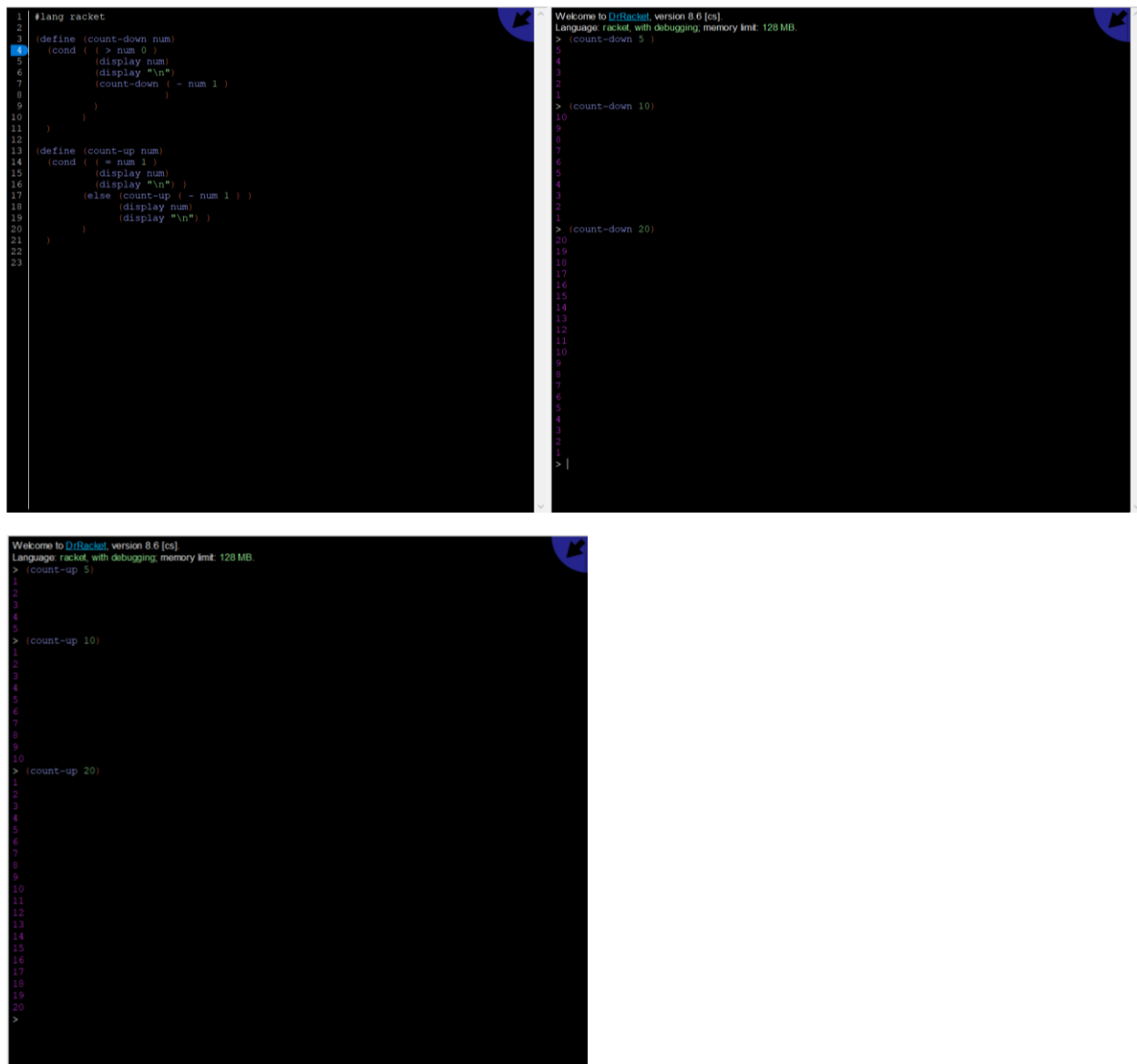
Timothy Astacio

CSC 344 Programming Languages

Racket Assignment 3 – Recursion in Racket

Learning Abstract: This assignment aims to explore the concept of recursion and its different applications in Racket. This assignment allowed me to gain a deeper understanding of the practices of recursion and it helped me develop practical skills in Racket.

Task 1 Count down and up Code and Demo:



```
1 #lang racket
2
3 (define (count-down num)
4   (cond ( ( > num 0 )
5         (display num)
6         (display "\n")
7         (count-down (- num 1))
8       )
9   )
10 )
11
12
13 (define (count-up num)
14   (cond ( ( = num 1 )
15         (display num)
16         (display "\n")
17         (else (count-up (- num 1) )
18               (display num)
19               (display "\n") )
20       )
21   )
22 )
23
```

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging, 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
> |

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging, memory limit: 128 MB.

> (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 and Demo:

```
1 #lang racket
2
3 (define (triangle-of-stars num)
4   (cond ( (= num 0)
5           (display "") )
6         (else (triangle-of-stars (- num 1))
7               (displayln (string-join (make-list num "*") " ") ) )
8       )
9   )
10
```

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging, memory limit: 128 MB.

```
> (triangle-of-stars 5)
*
* *
* * *
* * * *
* * * * *
```

```
> (triangle-of-stars 0)
> (triangle-of-stars 15)
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
```

Task 3 Flip Coin Code and Demo:

```

1 #lang racket
2
3 (define (flip-for-difference num)
4   (define (flip)
5     (if (= (random 2) 0) 'h 't))
6   )
7
8   (define (flip-helper num currentNum)
9     (define negate (* num -1))
10    (cond (
11      (and (< currentNum num) (> currentNum negate))
12      (let ((value (flip)))
13        (display (if (eq? value 't) "t" "h"))
14        (flip-helper num (if (eq? value 't) (- currentNum 1) (+ currentNum 1))
15          )
16      )
17    )
18    (else (display "**")))
19  )
20 )
21
22 (flip-helper num 0)
23
24
25
26

```

```

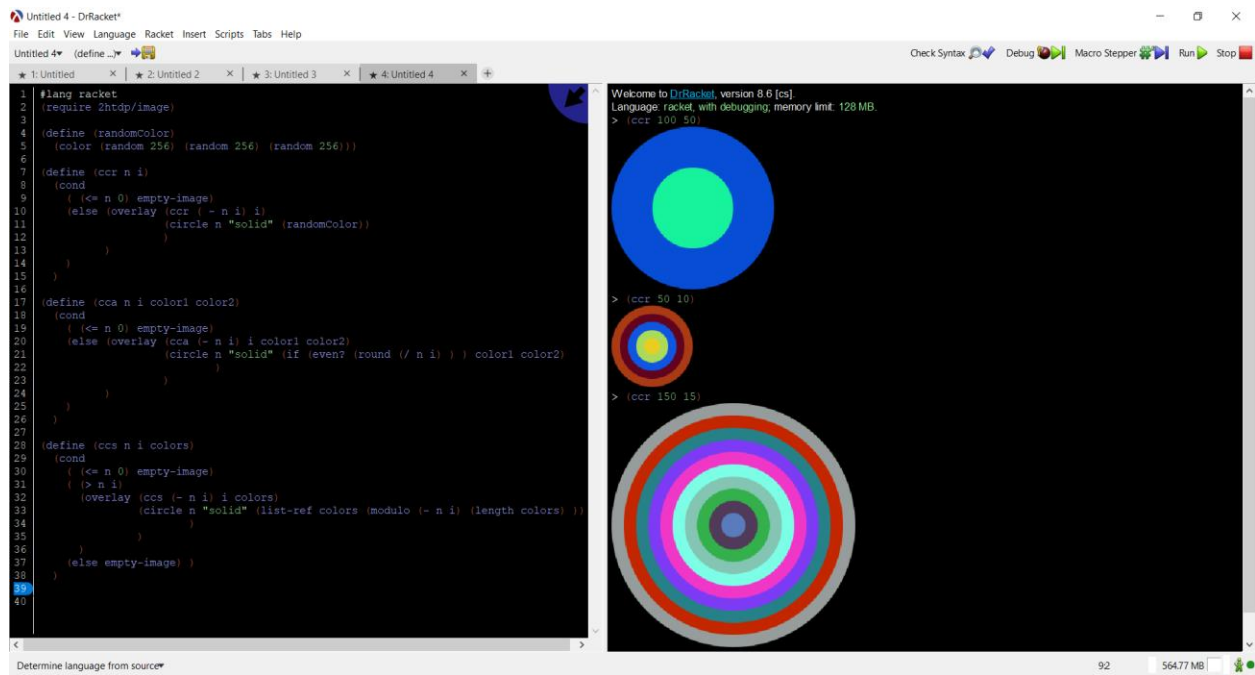
> (flip-for-difference 1)
h
> (flip-for-difference 1)
h
> (flip-for-difference 1)
h
> (flip-for-difference 1)
h
> (flip-for-difference 2)
hhhh
> (flip-for-difference 2)
shhtchhhhhh
> (flip-for-difference 2)
hhhh
> (flip-for-difference 2)
tt
> (flip-for-difference 2)
shhtt
> (flip-for-difference 2)
shhchshah
> (flip-for-difference 3)
hhhtthttthcthhhhhtthtthtttt
> (flip-for-difference 3)
hthttt
> (flip-for-difference 3)
shthhththhcthhhh
> (flip-for-difference 3)
ttt
> (flip-for-difference 3)
hhh
> (flip-for-difference 3)
shhchtthtt
> (flip-for-difference 4)
tthttt
> (flip-for-difference 4)
tthhctththctthct
> (flip-for-difference 4)
hhhtcthhthttttt
> (flip-for-difference 4)
hthctthttthcthhhhhtthtthtttt
> (flip-for-difference 4)
hhthhhh
> (flip-for-difference 4)

```

```
> flip-for-difference 4)
ctthct
> (flip-for-difference 4)
ctthctctthctthctt
> (flip-for-difference 4)
hhthctthctttttt
> flip-for-difference 4)
hththctttthctthhhthctthctthctt
> (flip-for-difference 4)
hththh
> (flip-for-difference 4)
hthctctthhh
> (flip-for-difference 4)
ctththhctttt
> (flip-for-difference 4)
ctththhhctthctthh
> |
```

Task 4 Laying down Colorful Concentric Disks Code and Demo:

CCR DEMO AND CODE:

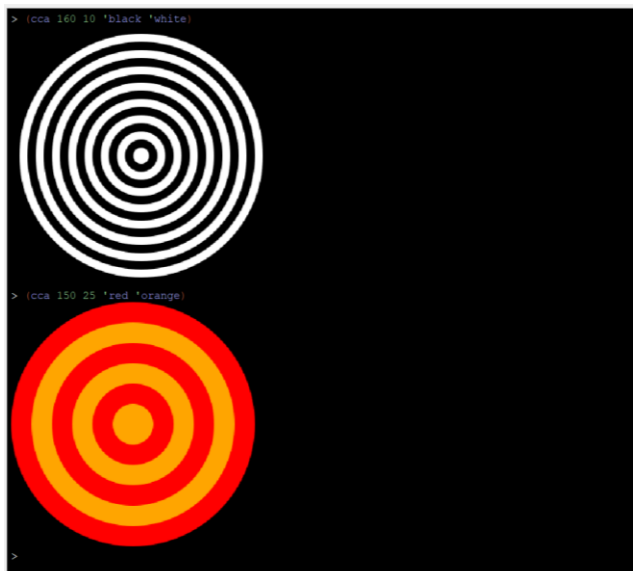


The screenshot shows the DrRacket IDE with a Racket script. The code defines three functions: `ccr`, `cca`, and `ccs`. `ccr` creates a circle with a random color. `cca` creates a circle with a specific color. `ccs` creates a circle with a list of colors. The output window shows the results of these functions: `(ccr 100 50)` produces a blue circle with a green center; `(cca 50 10)` produces a small yellow circle with a red border; `(ccr 150 15)` produces a large circle with a complex, multi-colored pattern.

```
1 #lang racket
2 (require 2htdp/image)
3
4 (define (randomColor)
5   (color (random 256) (random 256) (random 256)))
6
7 (define (ccr n i)
8   (cond
9     [(<= n 0) empty-image]
10    [else (overlay (ccr (- n 1) i)
11                   (circle n "solid" (randomColor)))]
12  ))
13
14
15
16
17 (define (cca n i color1 color2)
18   (cond
19     [(<= n 0) empty-image]
20    [else (overlay (cca (- n 1) i color1 color2)
21                   (circle n "solid" (if (even? (round (/ n i))) color1 color2)))]
22  ))
23
24
25
26
27
28 (define (ccs n i colors)
29   (cond
30     [(<= n 0) empty-image]
31     [(> n i)
32      (overlay (ccs (- n 1) i colors)
33               (circle n "solid" (list-ref colors (modulo (- n i) (length colors)))))]
34     [else empty-image]]
35  ))
36
37
38
39
40
```

92 564.77 MB

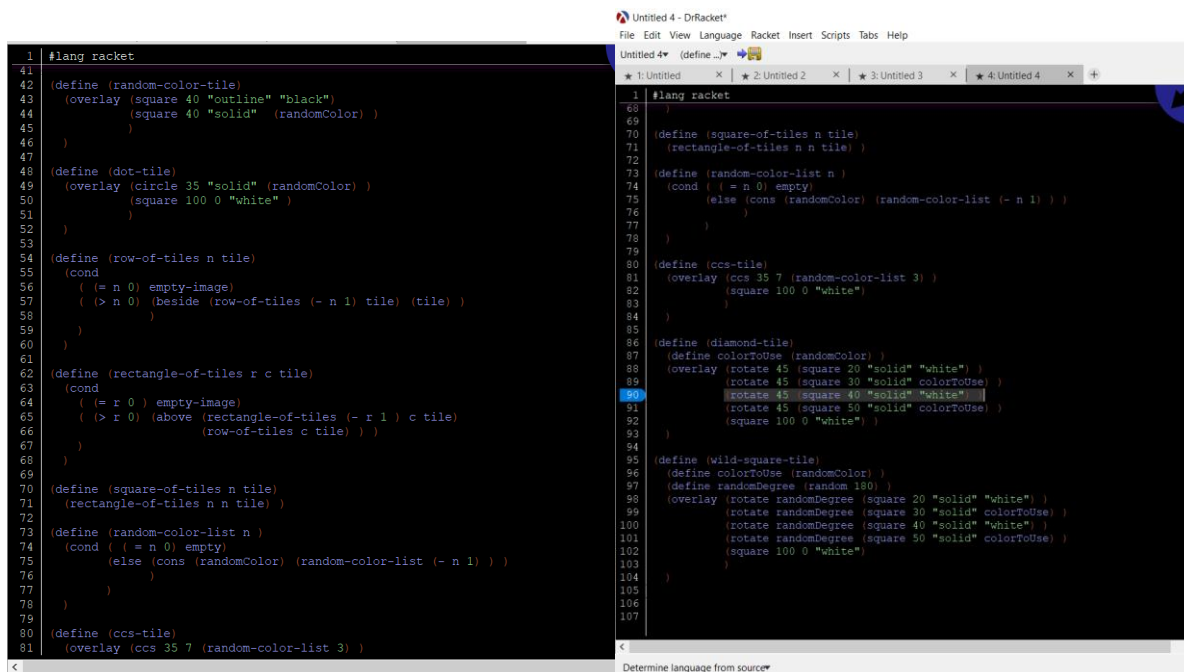
CCA AND CCS DEMOS:





Task 5 Variations on Hirst Dots Code and Demos:

Code:



DEMOS:

