

```

1 #lang racket
2 ;Darius Hooks
3 ;Nov 15, 2016
4 ;Project #2
5 ;
6 ;
7 ;
8 ;;;;;;;;;;;;;;;;;;;;;;;;;;
9 ;;;DEGREE OF POLYNOMIAL;;;
10 ;;;;;;;;;;;;;;;;;;;;;;;;;;
11 (define degree
12   (lambda (ply)
13     (if(null? ply)
14       0
15       (- (length ply) 1)
16     )
17   )
18 )
19 ;
20 ;
21 ;
22 ;;;;;;;;;;;;;;;;;;;;;;;;;;
23 ;;;DISPLAY TERM;;;;;;
24 ;;;;;;;;;;;;;;;;;;;;;;;;;;
25 (define displayTerm
26   (lambda (coef exp)
27     (cond
28       ;IF COEFFICIENT IS 1
29       [(= coef 1) (begin (if (= exp 0) (begin(display coef)(newline)) (if (= exp 1)
29 (begin(display "x")(newline)) (begin (display "x^")(display exp)))))]
30
31       ;IF COEFFICIENT IS GREATER THAN 1
32       [(> coef 1) (begin (display coef) (if (= exp 0) (newline) (if (= exp 1)
32 (begin(display "x")(newline)) (begin (display "x^")(display exp)))))]
33
34       ;IF COEFFICIENT IS 0
35       [(= coef 0) (display "Not a polynomial")]
36     );END COND
37   )
38 )
39 );END DISPLAY TERM
40 ;
41 ;
42 ;
43 ;;;;;;;;;;;;;;;;;;;;;;;;;;
44 ;;;DISPLAY POLYNOMIAL;;;
45 ;;;;;;;;;;;;;;;;;;;;;;;;;;
46 (define displayPoly
47   (lambda (p)
48     (cond

```

```

62         );END BEGIN
63     );END
64
65     ;IF FRONT OF LIST NOT 0
66     ((not (equal? (car p) 0))
67     (begin
68         (cond
69             ((equal? (- (length p) 1) 1) (begin (if (equal? (car p) 1) (display
69 "x") (begin(display (car p)) (display "x")))) (if (equal? (cadr p) 0) (displayPoly (cdr
69 p)) (begin (display " + ") (displayPoly (cdr p))))))
70             ((equal? (- (length p) 1) 0) (begin (display (car p)) (displayPoly (cdr
70 p))))
71             (else (if (equal? (car p) 1) (display "x^") (begin(display (car p))
71 (display "x^")))) (display (- (length p) 1)) (if (equal? (cadr p) 0) (displayPoly (cdr
71 p)) (begin (display " + ") (displayPoly (cdr p))))))
72         );END COND
73     );END BEGIN
74 );END
75
76 );END COND
77 )
78 );END DISPLAY POLY
79 ;
80 ;
81 ;
82 ;;;;;;;;;;;;;;;;;;;;;;;;;;;
83 ;;;;;;;;;;isPOLYNOMIAL;;;;;;;;;
84 ;;;;;;;;;;;;;;;;;;;;;;;;;;;
85 (define polynomial?
86     (lambda (z)
87         (let ([num 0])
88             (counter z num)
89         )
90     )
91 )
92 ;HELPER FUNCTION FOR POLYNOMIAL
93 (define counter
94     (lambda (lst i)
95         (cond
96             ((null? lst) #f)
97             ((string? lst) #f)
98             ((string? (car lst)) #f)
99             ((list? (car lst)) #f)
100             ((equal? (list-ref lst 0) 0) (if (= (length lst) 1) #t (if (equal? i 0) #f
100 (counter (cdr lst) (+ i 1)))))
101             ((= (length lst) 1) #t)
102             (else (counter (cdr lst) (+ i 1)))
103         )
104     )
105 )

```

```

119         (let ((coef (car p)) (x v) (expn (- (length p) 1)))
120             (+ (* coef (expt x expn)) (evalPoly (cdr p) v))
121         )
122     );END ELSE
123 );END COND
124 )
125 );END EVALUATE POLYNOMIAL
126 ;
127 ;
128 ;
129 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
130 ;;;;MULTIPLY POLYNOMIAL BY CONSTANT;
131 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
132 (define multiplyPolyByConstant
133     (lambda (p a)
134         (cond
135             ((equal? a 0) '(0))
136             (else (map (lambda (x) (* x a)) p))
137         )
138     )
139 )
140 ;
141 ;
142 ;
143 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
144 ;;;;MULTIPLY POLYNOMIAL BY X;
145 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
146 (define multiplyPolyByX
147     (lambda (p)
148         (foldl cons '(0) (reverse p))
149     )
150 )
151 ;
152 ;
153 ;
154 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
155 ;;;;ADD POLYNOMIAL;
156 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
157 (define addPoly
158     (lambda (p1 p2)
159         (cond
160             ;LENGTH P1 > LENGTH P2
161             ((> (length p1) (length p2))
162              (begin
163                  (let [(p3 (append (make 0 (- (length p1) (length p2))) p2))]
164                      (let ([p (map (lambda (x y) (+ x y)) p1 p3)])
165                          (cond
166                              ((equal? (foldl + 0 p) 0) '(0))
167                              ((equal? (car p) 0) (begin (remove 0 p) p))
168                              (else p)

```

```

182         ((equal? (car p) 0) (begin (remove 0 p) p))
183         (else p)
184     )
185 )
186 )
187 )
188 );END 2ND CONDITION
189
190 ;LENGTH P1 = LENGTH P2
191 ((= (length p1) (length p2))
192  (let ([p (map (lambda (x y) (+ x y)) p1 p2)])
193      (cond
194          ((equal? (foldl + 0 p) 0) '(0))
195          ((equal? (car p) 0) (begin (remove 0 p) p))
196          (else p)
197      )
198  )
199 )
200 ;END 3RD CONDITION
201 )
202 )
203 )
204 ;
205 ;
206 ;
207 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
208 ;;;;;;;;;SUBTRACT POLYNOMIAL;;;;;;;;
209 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
210 (define subtractPoly
211   (lambda (p1 p2)
212     (cond
213       ;LENGTH P1 > LENGTH P2
214       ((> (length p1) (length p2))
215        (begin
216          (let [(p3 (append (make 0 (- (length p1) (length p2))) p2))]
217            (let ([p (map (lambda (x y) (- x y)) p3 p1)])
218              (cond
219                  ((equal? (foldl + 0 p) 0) '(0))
220                  ((equal? (car p) 0) (begin (remove 0 p) p))
221                  (else p)
222              )
223            )
224          )
225        )
226       );END 1ST CONDITION
227
228 ;LENGTH P2 > LENGTH P1
229 ((< (length p1) (length p2))
230  (begin
231    (let [(p3 (append (make 0 (- (length p2) (length p1))) p1))]

```

```

245     (let ([p (map (lambda (x y) (- x y)) p1 p2)])
246       (cond
247         ((equal? (foldl + 0 p) 0) '(0))
248         ((equal? (car p) 0) (begin (remove 0 p) p))
249         (else p)
250       )
251     )
252   )
253 ;END 3RD CONDITION
254 )
255 )
256 )
257 ;
258 ;
259 ;
260 ;
261 ;HELPER FUNCTION FOR ADD AND SUBTRACT
262 (define make
263   (lambda (n i)
264     (if (= i 0)
265         '()
266         (cons n (make n (- i 1)))
267     )
268   )
269 )
270 ;
271 ;
272 ;
273 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
274 ;;;;;;;;;;;MULTIPLY POLYNOMIAL;;;;;;;;;;
275 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
276 (define multiplyPoly
277   (lambda (p1 p2)
278     (let ([x "INCOMPLETE FUNCTION"])
279       (display x)
280       '()
281     )
282   )
283 )
284 ;
285 ;
286 ;
287 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
288 ;;;;;;;;;;;TEST CASES;;;;;;;;;;
289 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
290 (define testcases
291   (lambda ()
292     (let ( [t1 '( 2 6 0 4 3 2)]
293           [t2 '(6 -3 0 -3 0 )]
294           [t3 '( 1 -3 0 0 0 4 -12)] )

```

```
308
309 (displayln (evalPoly t1 -2))
310 (displayln ( evalPoly t2 1.50))
311 (displayln (evalPoly t3 3))
312
313 (displayln (polynomial? t3))
314 (displayln( polynomial? "polynomial"))
315 (displayln ( polynomial? '( 0 5 6 7) ) )
316 (displayln (polynomial? (subtractPoly t1 t1)))
317 )))
```