#lang racket

;과제1 Integrating any function

```
(define (integral func num-steps x1 x2)
  (if (= num-steps 0) 0
      (+ (abs (* (func x1) (/ (- x2 x1) num-steps)))
         (integral func (- num-steps 1) (+ x1 (/ (- x2 x1) num-steps)) x2))))
```

;일단 기본 적분 하는 식은 이전과제 에서 재귀 함수를 응용해 만들었다

;이전과제 에서는 함수가 지정되어 있었지만 여기서는 함수를 사용자가 직접 넣을 수 있다.

;그래서 넓이를 구하는 부분에서 func를 부르는 부분이 다르다.

;과제1 test case:

```
> (integral (lambda (x) (expt x 2)) 1 3 5)
(integral (lambda (x) (expt x 2)) 2 3 5)
18
25
>
```

(integral (lambda (x) (expt x 2)) 1 3 5)
;x제곱 함수에서 3~5까지 사각형의 개수가 1개일 때 넓이 18

(integral (lambda (x) (expt x 2)) 2 3 5)
;x제곱 함수에서 3~5까지 사각형의 개수가 2개일 때 넓이 25

;과제2 Area of a unit circle

```
(define (approx-pi num-steps)
  (* (integral (lambda (x) (sqrt (- 1 (* x x)))) num-steps 0 1) 4))
```

;루트를 표현하는 특수 프로시저는 sqrt 다.

;반지름이 1인 원의 함수는 x^2+y^2=1 이고 y에 관한 식으로 바꾸면 +-루트(1-x^2)이다.

;그러므로 x,y가 양수인 부분의 넓이에 4를 곱하게 되면 원 전체의 넓이가 나오게 된다.

;그러므로 **과제1에서 만든 프로시저의 func 부분에 루트(1-x^2)를 넣으면 된다.**

;원의 넓이를 구하는 공식은 πr^2 이고 이 원에서의 반지름은 1이므로 넓이는 π 이다.

;하지만 우리가 만든 프로시저는 사각형으로 적분하여 근삿값을 찾는 프로시저 이므로 num-steps가 커지면 3.14에 가까워진다.

;과제2 test case:

```
> (approx-pi 1)
(approx-pi 1000)
4
3.143555466911027
>
```

(approx-pi 1) ;사각형이 1개일 때 넓이 4

(approx-pi 1000) ;사각형이 1000개일 때 넓이 3.1435 ... π 와 매우 비슷하다.

;과제3 Integrating with pieces of any shape

```
(define (rectangle func x1 x2)
  (abs (* (- x2 x1) (func x1))))
```
;func 함수에서의 x1에서 x2까지의 사각형의 넓이를 구하는 함수.

```
(define (trapezoid func x1 x2)
  (abs (/ (* (+ (func x1) (func x2)) (- x2 x1)) 2)))
```
;func 함수에서의 x1에서 x2까지의 사다리꼴의 넓이를 구하는 함수.

```
(define (integral-with piece func num-steps x1 x2)
  (if (= num-steps 0) 0 (+ (piece func x1 (+ x1 (/ (- x2 x1) num-steps)))
    (integral-with piece func (- num-steps 1) (+ x1 (/ (- x2 x1) num-steps)) x2))))
```

;사다리꼴 형식으로 넓이를 구할지, 직사각형으로 넓이를 구할지 piece에서 판단하고 num-step로 사각형의 개수를 받아 적분하는 프로시저이다.

;piece 에서 rectangle 또는 trapezoid로 인자들을 넘길 때, 이 함수들은 num-steps가 없으므로 x1은 그대로 넘기고 x2는 x1~x2 길이를 사각형의 개수로 나눈 값을 x1에 더한 값으로 넘겨준다.

;그 후 재귀적으로 자기 자신을 부르고 num-steps는 1을 빼서, x1은 x1~x2 길이를 사각형의 개수로 나눈 값을 x1에 더한 값, x2는 그냥 넘겨주게 되고, num-steps가 0이 되면 0을 반환 하고 지금까지 리턴된 값들을 모두 더한다.

;과제3 test case:

```
> (integral-with rectangle (lambda (x) (expt x 2)) 2 1 3)
(integral (lambda (x) (expt x 2)) 2 1 3)
(integral-with trapezoid (lambda (x) (expt x 2)) 2 1 3)

5
5
9
> |
```

(integral-with rectangle (lambda (x) (expt x 2)) 2 1 3)
;사각형으로 함수의 넓이를 구했을 때, 5

(integral (lambda (x) (expt x 2)) 2 1 3)
;처음 정의한 프로시저로 같은 함수의 넓이를 구했을 때, 5 같다.

(integral-with trapezoid (lambda (x) (expt x 2)) 2 1 3)
;사다리꼴로 함수의 넓이를 구했을 때, 9

;과제4 Better approximation of π

```
(define (better-pi num-steps)
  (* (integral-with trapezoid (lambda (x) (sqrt (- 1 (* x x)))) num-steps 0 1) 4))
```

;더 정확한 π값을 찾는 프로시저, integral-with의 piece에 trapezoid를 넣고, func 부분에 (lambda (x) (sqrt (- 1 (* x x))))를 넣은 후 사각형 개수와 범위를 지정해준다.

;과제4 test case:

```
> (better-pi 1)
(better-pi 1000)
2
3.1415554669110293
>
```

(better-pi 1) ;사다리꼴 개수가 1개일 때 넓이 2 (사각형일 때 넓이 4)
(better-pi 1000) ;사다리꼴 개수가 1000개일 때 넓이 3.1415... (사각형일 때보다 좀 더 정확하다.)

;Part 2

```
(define (deriv-constant wrt constant) 0)
```

;숫자를 wrt에 관해 미분했을 때 나오는 값은 항상 0이다.

;과제5 Derivative of a variable

```
(define (deriv-variable wrt var)
  (if (eq? wrt var) 1 0))
```

;wrt로 var을 미분했을 때 나오는 값 1, 그외엔 0

;과제5 test case:

```
> (deriv-variable 'x 'x)
(deriv-variable 'x 'y)
(deriv-variable 'y 'y)
1
0
1
>
```

(deriv-variable 'x 'x) ;1
(deriv-variable 'x 'y) ;0

(deriv-variable 'y 'y) ;1

;과제6 Calling the right function

```
(define (sum? expr)
  (if (eq? '+ (car expr)) #t #f))
```
;expr이 덧셈을 하는 식인지를 판단하여 참, 거짓을 반환한다.

```
(define (product? expr)
  (if (eq? '* (car expr)) #t #f))
```
;expr이 곱셈을 하는 식인지를 판단하여 참, 거짓을 반환한다.

```
(define (derivative wrt expr)
  (cond ((number? expr) (deriv-constant wrt expr)) ;expr이므로 0 반환.
        ((symbol? expr) (deriv-variable wrt expr)) ;wrt로 expr을 미분한 값.
        ((sum? expr) (deriv-sum wrt expr)) ;expr이 덧셈을 하는 식일 때 deriv-sum을 호출.
        ((product? expr) (deriv-product wrt expr)) ;expr이 곱셈을 하는 식일 때, deriv-product를 호출.
        (else (error "Don't know how to differentiate" expr)))) ;위의 예시 이외에 다른 것이 들어오면 오류 출력.
```

;과제6 test case:

```
> (derivative 'x 1)
(derivative 'x 'x)
0
1
> |
```
(derivative 'x 1) ;0
(derivative 'x 'x) ;1

;과제7 Derivative of a sum

```
(define (deriv-sum wrt expr)
  (let ((first (car expr))
        (other (cdr expr)))
    (if (eq? first '+) (display "(+") null)
    (if (null? other) (display ")₩n")
        (begin (display " ") (display (derivative wrt (car other))) (deriv-sum wrt other)))))
```

;deriative에서 expr이 덧셈일 때 호출되는 함수이다. (과제7번을 하기 전에 이 내용이 과제6번의 (?sum expr) 뒤에 들어가 있었다.)
;처음에 first에 +가 들어있을 때만 +를 출력하고 그 이후로는 +가 출력 되지 않음
;begin을 이용해 other가 null이 아닐 때 할 행동들을 모아서 실행되게 하였다.
;other가 null이 아닐 때 other의 car를 미분하여 출력하고 자기 자신을 호출한다.
;마지막에 other가 null이면 괄호를 닫아 출력 한다.
;(과제 9)그리고 재귀를 이용해 덧셈 식에 몇 개의 인자가 있던지 모두 계산할 수 있게 했다.

;과제7 testcase:

```
> (derivative 'x '(+ x 9))
(deriv-sum 'x '(+ 9 x))
(+ 1 0)
(+ 0 1)
>
```

(derivative 'x '(+ x 9))
;(+ 1 0) derivative를 썼을 때 덧셈인지 판단했으므로 deriv-sum이 호출되어 미분된 값이 출력됨.

(deriv-sum 'x '(+ 9 x))
;(+ 0 1) deriv-sum으로 호출했을 때, 입력 내용을

;과제8 Derivative of a product

```
(define (deriv-product wrt expr)
  (let ((first (cadr expr))
        (last (caddr expr)))
    (display "(+ (* ")
    (display first)
    (display " ")
    (display (derivative wrt last))
    (display ") (* ")
    (display (derivative wrt first))
    (display " ")
    (display last)
    (display "))₩n")))
```

;derivative 에서 expr이 곱셈일때 호출되는 함수이다. 과제8번을 하기전에 이 내용이 (product? expr) 뒤에 들어가있었다.
;곱셈의 미분은 (x * y)' 라면 ((x' * y)+(x * y'))이므로 expr이 곱셈일때 이 형식으로 출력되게 하는 함수이다.

;과제8 test case:

```
> (derivative 'x '(* x 9))
(deriv-product 'x '(* 9 x))
(+ (* 1 9) (* x 0))
(+ (* 0 x) (* 9 1))
>
```

(derivative 'x '(* x 9))
;(+ (* 1 9) (* x 0)) expr이 곱셈이므로 deriv-product가 호출되고, 그때의 미분값이 출력된다.

(deriv-product 'x '(* 9 x))
;(+ (* 0 x) (* 9 1)) 식의 순서를 바꿔서 입력하면 반대로 나온다.

;과제9 Additional testing

;식에 두개 이상의 요소가 있어도 출력할 수 있게 전의 코드를 수정하기.

;과제9 test case:

```
> (derivative 'x '(+ x 9 x))
(deriv-sum 'x '(+ x 9 x 1 5 x))
(+ 1 0 1)
(+ 1 0 1 0 0 1)
> |
```

(derivative 'x '(+ x 9 x)) ;(+ 1 0 1) 출력

(deriv-sum 'x '(+ x 9 x 1 5 x)) ;(+ 1 0 1 0 0 1) 출력