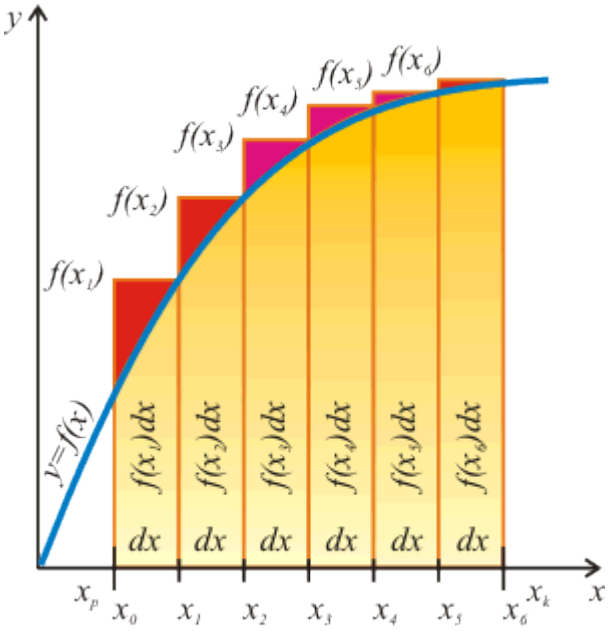
Zadanie 5.1

(*) Zaimplementuj funkcję **cubeRoot** wyznaczającą pierwiastek trzeciego stopnia z liczby rzeczywistej. Jeżeli `y` jest przybliżeniem pierwiastka trzeciego stopnia z `x`, to kolejne (lepsze) przybliżenie uzyskujemy ze wzoru `1/3(2y + x/(y^2))`. Wynik obliczeń jest satysfakcjonujący jeśli `y^3` różni się od `x` nie więcej niż założona dokładność. Funkcję umieść w module **Math**.

Wskazówka: Na slajdzie 30 zaprezentowano analogiczne rozwiązanie do wyznaczania pierwiastka kwadratowego.

Zadanie 5.2



Utwórz moduł **Integration** i zamieść w nim poniższy kod. Uzupełnij kod funkcji **rectangleRule** wyznaczającej przybliżoną wartość całki oznaczonej metodą prostokątów.

```
rectangleRule :: (Float -> Float) -> Float -> Float -> Int -> Float
rectangleRule (f) a b n = {- brakujący kod -}
  where
    h = (b - a) / fromIntegral(n)
    points = [a + fromIntegral(i) * h | i <- [1 .. n]]
```

- `f` - funkcja podcałkowa,
- `a`, `b` - końce przedziału całkowania,
- `n` - liczba przedziałów na które dzielimy przedział całkowania.

Przybliżoną wartość całki wyznacza się ze wzoru:

$$\sum_{i=1}^{n} h f(x_i)$$

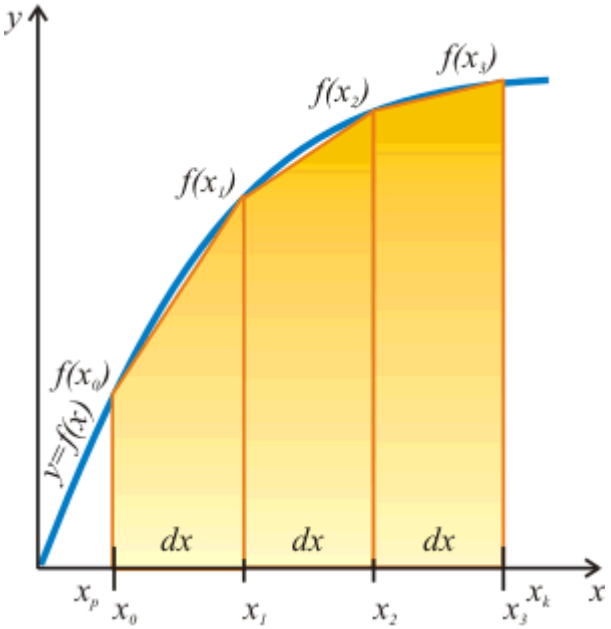
Przetestuj działanie funkcji rectangleRule wyznaczając całkę z funkcji:

- `sin` na przedziale (0, pi) - dokładna wartość 2

- `sqrt` na przedziale (0,1) - dokładna wartość 2/3
- `x^2` na przedziale (0,1) - dokładna wartość 1/3
- `x^3 + 2x` na przedziale (0,2) - dokładna wartość 8

W dwóch ostatnich przypadkach użyj kolejno **sekcji** i **notacji lambda**, aby zadać funkcję podcałkową.

Zadanie 5.3



(*) Zdefiniuj funkcję **trapezoidalRule** wyznaczającą przybliżoną wartość całki oznaczonej metodą trapezów. Przybliżoną wartość całki wyznacza się ze wzoru:

$$\frac{h}{2}(f(a)+f(b))+\sum_{i=1}^{n-1} hf(x_i)$$

Funkcję umieść w module **Integration**. Przetestuj działanie funkcji **trapezoidalRule** dla przykładów z zadania 2 i porównaj wyniki.

Zadanie 5.4

W module **Trees** umieść poniższą definicję drzewa binarnego:

```
data Tree a = Leaf a
            | Node a (Tree a) (Tree a)
            | Null
```

Trzeci konstruktor **Null** został użyty jako **pusty węzeł**, do oznaczenia braku lewego lub prawego poddrzewa.

1. Przedefiniuj funkcję **treeSize** ze slajdu 57 tak, aby uwzględniała zmienioną definicję typu **Tree**.
2. Zaimplementuj funkcje **showTree** generującą tekstową reprezentację drzew binarnych zgodną w podanymi przykładami:

```
showTree :: (Show a) => Tree a -> String


bt1 = Node 7 (Node 4 (Leaf 2) (Leaf 5)) (Leaf 10)


bt2 = Node 7 (Node 4 (Leaf 2) (Leaf 5))
             (Node 10 (Leaf 9) (Node 13 (Leaf 11) (Leaf 15)))


bt3 = Node 7 (Leaf 1) Null


-- showTree bt1
-- "7 L(4 L(2) R(5)) R(10)"
-- showTree bt2
-- "7 L(4 L(2) R(5)) R(10 L(9) R(13 L(11) R(15)))"
-- showTree bt3
-- "7 L(1) R(())"
```

Zadanie 5.5

W module **Trees** zaimplementuj funkcję dodającą nowy element do drzewa. Zakładamy, że drzewo będące argumentem jest uporządkowane i dodanie elementu nie psuje porządku (elementy w lewym poddrzewie są mniejsze, a w prawym większe od wartości w węźle). Jeżeli w drzewie istnieje już element o danej wartości, to do drzewa nie dodajemy nowego węzła.

```
add :: (Ord a) => a -> Tree a -> Tree a
```

Zadanie 5.6

1. (*) Do modułu **Trees** dodaj funkcję **elemTree** ze slajdu 57. Popraw funkcję tak, aby uwzględniała konstruktor **Null**.
2. (*) W module **Trees** zaimplementuj funkcję **countLeaves** zwracającą informację o liczbie liści w drzewie.

Zadanie 5.7

1. (*) W module **Trees** zaimplementuj funkcję **tree2list** przekształcającą drzewo na uporządkowaną listę wartości.
2. (*) W module **Trees** zaimplementuj funkcję **list2tree** przekształcającą uporządkowaną listę na zrównoważone drzewo binarne - wysokość prawego i lewego poddrzewa dla każdego węzła mogą się różnić co najwyżej o 1.