Зад.1. Нека е дадена едноместна функция **f** и списък от възможни нейни аргументи. Напишете функции **argmax** и **argmin**, които намират за кой елемент на списъка **f** достига съответно максимална и минимална стойност. Ако най-високата (съответно най-ниската) стойност се достига за повече от един елемент от списъка, нека функциите връщат *първия от тези елементи*. При празен подаден списък изберете подходяща "специална" стойност на връщане.

Примери:

```
(argmax (lambda (x) (* x x)) '(1 3 0 4 2.5 -4)) -> 4 ; а не -4
(argmin length '((1 2) () (2 a 5 7) (2 4))) -> '()
```

Заб.: В racket има вградени **argmin** и **argmax**, които *(очевидно)* са забранени за тази задача.

Зад.2. Нека е дадено неотрицателно цяло число **n**. Напишете функция **(reduce n)**, която го "редуцира" до едноцифрено по следната процедура:

- намира най-голямата цифра в числото и я "премахва" от него (при повече от едно срещания премахва най-лявата такава цифра)
- умножава новополученото число по тази премахната цифра и, ако полученото число не е едноцифрено, повтаря процедурата наново за него.

Нека, например, **n**=26364. Най-голямата цифра е 6 и след премахване на първата шестица получаваме 2364. Умножаваме 2364*6=14184, което още не е едноцифрено, така че продължаваме от това число.

Примери:

```
(reduce 9) -> 9
(reduce 27) -> 4
(reduce 757) -> 5
(reduce 1234) -> 8
(reduce 26364) -> 8
(reduce 432969) -> 0
(reduce 1234584) -> 8
(reduce 91273716) -> 6
```

Заб.: <u>За тази задача</u> е забранено използването на вградени функции от типа на **number->string**, **number->list**, **char->integer** и т.н.

Зад.3. Нека са дадени едноместна числова функция **f** и две числа **a** и **b** (a < b) такива, че:

- **f** е дефинирана и диференцируема в целия интервал **[a;b]**

- f(a)*f(b)<0

- **f** има точно един корен в интервала **(a;b)** (x∈(a;b) такова, че f(x)=0).

За намирането на този корен с точност до **eps** можем да използваме следните итерационни числени методи:

1. Двоично търсене:

- нека на дадена итерация сме локализирали корена до интервала $[a_i;b_i]$.

- изчисляваме стойността на **f** в средата на интервала mid = $(a_i+b_i)/2$ и гледаме получената стойност

- ако тя е достатъчно близо до 0*, значи сме намерили достатъчно добро приближение на корена и можем да приключим търсенето

- в противен случай знакът на тази стойност със сигурност съвпада със знака на **f** или в a_i, или в b_i (но не и в двете)

- отхвърляме тази половина на интервала с край, съвпадащ по знак с f(mid) и продължаваме търсенето в оставащата половина. Например при f(x) = x(x-5) и интервал [1;7] имаме f(1)=-4, f(7)=14 и f(4)=-1 (тук 4=(1+7)/2). Тогава търсенето трябва да продължи в интервала [4;7], а не [1;4].

- когато дължината на интервала стане по-малка от **eps**, можем да върнем средата му като "достатъчно добра апроксимация" на корена. За първоначален интервал за търсене можем да вземем **[a;b]**.

* *Упътване:* напишете и използвайте помощна функция **(approx-zero? x)**, която проверява дали абсолютната стойност на числото **x** е по-малка от някакъв друг предварително определен eps.

Напишете функция **(find-root f a b eps)**, която при тези условия използва някой от гореописаните методи *по ваш избор* и връща наредена двойка от намерения корен и броя итерации, необходими за достигането му. Имайте предвид, че поради естеството на тези методи е възможно за някои функции и/или интервали алгоритъмът да не приключва - вие няма нужда да се притеснявате за това и/или да го проверявате.