



በጥቅም ላይ የዋለው የጥናት ዓላማ «የጥናት ዓላማው ለ ለ ጥናት ዓላማው የተመሰረተ» በጥናት ላይ የተመሰረተ ነው። በጥናት ላይ የተመሰረተ የጥናት ዓላማው ለጥናት ዓላማው የተመሰረተ ነው።

. « »

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

000000 00 .000000 000 0000000 000000 00 00000000000 00 000 00000 000 00 000 0000 0000 000000
 000000 0000 **defn** 0000000000 00 000000 0000 00 .000000 000000000 0000000 000000 00 000000 00 00000000
 0000 000 .000 (**def** foo (fn...)) 0000 0000000000 0000 000 0000000000 000000 00000000 00000000 0000000 000000
 0000 0000 00 .0000000000 00 00 00000 000000 00 00 00 00 **var** 000 **Defn** .000 00000 00 **fn** 000000000 0000000
 0000 000000 00 00 000000000 0 00000000 00 00000 000000 000000 00000 00000 000000000 000000 00 0000000 00 0000
 00000 0 0000 00000 0000000 000000 00 00 000000 00000 0000 000000000 00000000000000 .00000 00000 00 000000 00
 000000 000000 00 000000000 00 000000 00000 000000 00000 00000 000000 00000000000000 .00000 00000 00 000000 00
 00000 000000 00 000000000 00 000000 00000 000000 00000 0000 00 .000 000000 0000000 0000000 00 00000 00
 0000 00 000000 00 00 000000 000 00 00000 0000000 000000 00000 0000 .00000 000000 000000 000 00 000 0000000
 .000000 000000 00 00 00000 0000000 000000 000 0 0000000

(defn double-sum

[a b]

(* 2 (+ a b)))

.double-sum 함수는 두 개의 정수를 더한 후 그 결과를 2로 곱하여 반환합니다.

(defn double-subtraction

[a b]

(* 2 (- a b)))

double-sum 함수와 마찬가지로 double-subtraction 함수는 두 정수의 차를 2로 곱하여 반환합니다. 예를 들어, (double-subtraction 5 3)은 4를 반환합니다.

(defn double-operator

[f a b]

(* 2 (f a b)))

(double-operator + 3 1) ;; 8

(double-operator - 3 1) ;; 4

double-operator 함수는 두 정수와 하나의 연산자를 받아, 연산자를 사용하여 두 정수를 연산한 후 그 결과를 2로 곱하여 반환합니다.

Filter

Filter 함수는 주어진 콜백 함수를 사용하여 리스트의 요소를 필터링합니다. 콜백 함수는 각 요소를 인자로 받아 true 또는 false를 반환합니다. true를 반환하는 요소는 필터링된 리스트에 포함되고, false를 반환하는 요소는 제외됩니다.

Filter

Filter 함수는 주어진 콜백 함수를 사용하여 리스트의 요소를 필터링합니다. 콜백 함수는 각 요소를 인자로 받아 true 또는 false를 반환합니다. true를 반환하는 요소는 필터링된 리스트에 포함되고, false를 반환하는 요소는 제외됩니다.

- evenNumbers 함수는 numbers 리스트에서 짝수만 필터링합니다.
- numbers 리스트에서 짝수만 필터링합니다.
- evenNumbers 함수는 numbers 리스트에서 짝수만 필터링합니다.

.evenNumbers 함수는 numbers 리스트에서 짝수만 필터링합니다.

```
var numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
var evenNumbers = [];
```

```
for (var i = 0; i < numbers.length; i++) {
```

```
  if (numbers[i] % 2 == 0) {
```

```
    evenNumbers.push(numbers[i]);
```

```
  }
```

```
}
```

```
console.log(evenNumbers); // (6) [0, 2, 4, 6, 8, 10]
```

JavaScript 中 筛选数组 的 方法 有 `filter` 方法。它接收一个函数，返回一个由该函数返回 `true` 的元素的数组。原数组不会被改变。

```
(defn even-numbers
```

```
  [coll]
```

```
  (filter even? coll))
```

```
(even-numbers [0 1 2 3 4 5 6 7 8 9 10]) ;; (0 2 4 6 8 10)
```

在 Clojure 中，`filter` 函数接收一个谓词函数和一个集合，返回一个包含所有满足谓词函数的元素的集合。原集合不会被改变。

```
var filterArray = function(x, coll) {
```

```
  var resultArray = [];
```

```
  for (var i = 0; i < coll.length; i++) {
```

```
    if (coll[i] < x) {
```

```
      resultArray.push(coll[i]);
```

```
    }
```

```
}
```

```
return resultArray;
```

```
}
```

```
console.log(filterArray(3, [10, 9, 8, 2, 7, 5, 1, 3, 0])); // (3) [2, 1, 0]
```

在 JavaScript 中，`filterArray` 函数接收一个数字 `x` 和一个数组 `coll`，返回一个包含所有小于 `x` 的元素的数组。原数组不会被改变。

この関数は、与えられたリストから、指定した条件を満たす要素を抽出する。Clojure の filter 関数は、与えられた関数とリストを受け取り、関数の結果が true である要素を抽出する。

```
(defn filter-array
```

```
  [x coll]
```

```
  (filter #( > x %) coll))
```

```
(filter-array 3 [10 9 8 2 7 5 1 3 0]) ;; (2 1 0)
```

x が与えられた関数の引数として渡され、filter 関数は、与えられたリスト coll から、関数の結果が true である要素を抽出する。この例では、3 より大きい要素を抽出している。filter 関数は、与えられた関数とリストを受け取り、関数の結果が true である要素を抽出する。この例では、3 より大きい要素を抽出している。

```
(def people [{:name "TK" :age 26}
```

```
             {:name "Kaio" :age 10}
```

```
             {:name "Kazumi" :age 30}])
```

```
(defn over-age
```

```
  [people]
```

```
  (filter
```

```
    #( < 21 (:age %))
```

```
    people))
```

```
(over-age people) ;; ({:name "TK", :age 26} {:name "Kazumi", :age 30})
```

この関数は、与えられたリストから、指定した条件を満たす要素を抽出する。Clojure の filter 関数は、与えられた関数とリストを受け取り、関数の結果が true である要素を抽出する。この例では、21 より小さい要素を抽出している。filter 関数は、与えられた関数とリストを受け取り、関数の結果が true である要素を抽出する。この例では、21 より小さい要素を抽出している。

Map

Map は、キーと値のペアのコレクションである。Clojure の map 関数は、与えられた関数とリストを受け取り、関数の結果を新しいリストにマッピングする。この例では、people リストの各要素に対して、name は、age は、という形式で新しいリストを作成している。map 関数は、与えられた関数とリストを受け取り、関数の結果を新しいリストにマッピングする。この例では、people リストの各要素に対して、name は、age は、という形式で新しいリストを作成している。

```
var people = [
```

```
  { name: "TK", age: 26 },
```

```

{ name: "Kaio", age: 10 },

{ name: "Kazumi", age: 30 }

];

var peopleSentences = [];

for (var i = 0; i < people.length; i++) {

    var sentence = people[i].name + " is " + people[i].age + " years old";

    peopleSentences.push(sentence);

}

console.log(peopleSentences); // ['TK is 26 years old', 'Kaio is 10 years old', 'Kazumi is 30 years old']

```

: Clojure

```

(def people [{:name "TK" :age 26}

              {:name "Kaio" :age 10}

              {:name "Kazumi" :age 30}])

(defn people-sentences

  [people]

  (map

    #(str (:name %) " is " (:age %) " years old")

    people))

(people-sentences people) ;; ("TK is 26 years old" "Kaio is 10 years old" "Kazumi is 30 years old")

```

4- Clojure

```

var values = [1, 2, 3, -4, 5];

for (var i = 0; i < values.length; i++) {

    values[i] = Math.abs(values[i]);

}

console.log(values); // [1, 2, 3, 4, 5]

```

Math.abs

map 関数は、与えられた関数をリストの各要素に適用し、結果のリストを返します。この関数は、関数、リスト、および結果のリストの 3 つの引数で呼び出されます。関数は、リストの各要素に対して呼び出され、結果のリストに追加されます。map 関数は、関数、リスト、および結果のリストの 3 つの引数で呼び出されます。関数は、リストの各要素に対して呼び出され、結果のリストに追加されます。

```
(defn to-absolute
```

```
  [n]
```

```
  (if (neg? n)
```

```
    (* n -1)
```

```
    n))
```

```
(to-absolute -1) ;; 1
```

```
(to-absolute 1) ;; 1
```

```
(to-absolute -2) ;; 2
```

```
(to-absolute 0) ;; 0
```

map 関数は、与えられた関数をリストの各要素に適用し、結果のリストを返します。この関数は、関数、リスト、および結果のリストの 3 つの引数で呼び出されます。関数は、リストの各要素に対して呼び出され、結果のリストに追加されます。map 関数は、関数、リスト、および結果のリストの 3 つの引数で呼び出されます。関数は、リストの各要素に対して呼び出され、結果のリストに追加されます。

```
(defn update-list-map
```

```
  [coll]
```

```
  (map to-absolute coll))
```

```
(update-list-map []) ;; ()
```

```
(update-list-map [1 2 3 4 5]) ;; (1 2 3 4 5)
```

```
(update-list-map [-1 -2 -3 -4 -5]) ;; (1 2 3 4 5)
```

```
(update-list-map [1 -2 3 -4 5]) ;; (1 2 3 4 5)
```

map 関数は、与えられた関数をリストの各要素に適用し、結果のリストを返します。この関数は、関数、リスト、および結果のリストの 3 つの引数で呼び出されます。関数は、リストの各要素に対して呼び出され、結果のリストに追加されます。

Reduce

reduce 関数は、与えられた関数をリストの各要素に適用し、結果の値を返します。この関数は、関数、初期値、およびリストの 3 つの引数で呼び出されます。関数は、初期値に対して呼び出され、結果の値に追加されます。reduce 関数は、関数、初期値、およびリストの 3 つの引数で呼び出されます。関数は、初期値に対して呼び出され、結果の値に追加されます。

```
var orders = [
```

```

{ productTitle: "Product 1", amount: 10 },
{ productTitle: "Product 2", amount: 30 },
{ productTitle: "Product 3", amount: 20 },
{ productTitle: "Product 4", amount: 60 }
];

var totalAmount = 0;

for (var i = 0; i < orders.length; i++) {

  totalAmount += orders[i].amount;

}

console.log(totalAmount); // 120

```

上のコードは、orders 配列の amount 属性の値を合計 (amount sum) する。JavaScript では、Array オブジェクトの reduce 関数を使用して、この処理を簡潔に記述できる。

```

(defn shopping-cart

  [{ :product-title "Product 1" :amount 10 },

   { :product-title "Product 2" :amount 30 },

   { :product-title "Product 3" :amount 20 },

   { :product-title "Product 4" :amount 60 }])

(defn sum-amount

  [total-amount current-product]

  (+ (:amount current-product) total-amount))

(defn get-total-amount

  [shopping-cart]

  (reduce sum-amount 0 shopping-cart))

(get-total-amount shopping-cart) ;; 120

```

上記のコードでは、total-amount 変数に初期値 0 を設定し、sum-amount 関数を shopping-cart 配列の各要素に対して呼び出す。get-total-amount 関数は、reduce 関数を使用して、sum-amount 関数を shopping-cart 配列の各要素に対して呼び出す。reduce 関数は、Map オブジェクトを使用して、sum-amount 関数を呼び出す。

```

(defn shopping-cart

```

```
[{ :product-title "Product 1" :amount 10 },  
 { :product-title "Product 2" :amount 30 },  
 { :product-title "Product 3" :amount 20 },  
 { :product-title "Product 4" :amount 60 }])
```

```
(defn get-amount
```

```
  [product]
```

```
  (:amount product))
```

```
(defn get-total-amount
```

```
  [shopping-cart]
```

```
  (reduce + (map get-amount shopping-cart)))
```

```
(get-total-amount shopping-cart) ;; 120
```

20 30 10] 0000 000 00 .00000000 00 00 amount 00000 0000 0 00000 000000 00 00000 00 Get-amount
.00000 0000 000000 00 00000 000000 000 00000 00 reduce 00000 00 .00000 00 [60

00 00 00000 000000 0 0000 00000 00 000000 0000 00000 000000 0000 0000 000 000000 00 0000 000000
:00000 000000 0000000 00 000000 0000000 0000 0000 000 00 0000000 .0000 000000 00 0000

```
(def shopping-cart
```

```
  [{ :product-title "Functional Programming" :type "books"      :amount 10 },
```

```
    { :product-title "Kindle"                :type "eletronics" :amount 30 },
```

```
    { :product-title "Shoes"                  :type "fashion"    :amount 20 },
```

```
    { :product-title "Clean Code"             :type "books"      :amount 60 }])
```

00 000 000 00000 000000000 .00000 000 00 00 0000 000 0000000 00000 0000 000 00000 00000 000 00
:0000 000 000

- .0000000 00000 00 0000000 000
- .0000000 00000 00000 00 000000000 00 map 0000 00 0000 000
- .0000000 000 00 00 reduce 00 00000000 00 00 00000 00000

:000000 00000000000 000 000 00 000 0000000000

```
let shoppingCart = [
```

```
  { productTitle: "Functional Programming", type: "books", amount: 10 },
```

```
  { productTitle: "Kindle", type: "eletronics", amount: 30 },
```

```
  { productTitle: "Shoes", type: "fashion", amount: 20 },
```

```

    { productTitle: "Clean Code", type: "books", amount: 60 }
  ]

  const byBooks = (order) => order.type === "books";

  const getAmount = (order) => order.amount;

  const sumAmount = (acc, amount) => acc + amount;

  function getTotalAmount(shoppingCart) {

    return shoppingCart

      .filter(byBooks)

      .map(getAmount)

      .reduce(sumAmount, 0);

  }

  getTotalAmount(shoppingCart); // 70

```

📄 📄 📄

در این دوره، شما با مفاهیم پایه‌ای برنامه‌نویسی آشنا خواهید شد و به تدریج به سراغ مفاهیم پیشرفته‌تر خواهید رفت. در این دوره، شما با مفاهیم پایه‌ای برنامه‌نویسی آشنا خواهید شد و به تدریج به سراغ مفاهیم پیشرفته‌تر خواهید رفت. در این دوره، شما با مفاهیم پایه‌ای برنامه‌نویسی آشنا خواهید شد و به تدریج به سراغ مفاهیم پیشرفته‌تر خواهید رفت.

:📄 📄 📄

📄 📄 📄

:📄 📄 📄

[freecode camp](#)

[code mentor](#)

:📄 📄 📄

📄 📄 📄

📄 📄 📄

:📄 📄 📄

13:40 - 14/12/1398

:📄 📄 📄

📄 📄 📄 - 📄 📄 📄

📄 📄 📄

<https://www.shabakeh-mag.com/workshop/16637/%D8%A2%D8%B4%D9%86%D8%A7%DB%8C%D8%A8%D8%A7-%D9%85%D9%81%D9%87%D9%88%D9%85-%D8%AA%D9%88%D8%A7%D8%A8%D8%B9-%D8%AF%D8%B1%D8%AC%D9%87-%D8%A7%D9%88%D9%84-%D8%AF%D8%B1-%D8%A8%D8%B1%D9%86%D8%A7%D9%85%D9%87%E2%80%8C%D9%86%D9%88%DB%8C%D8%B3%DB%8C-%D8%AA%D8%A7%D8%A8%D8%B9%DB%8C>