Hàm trong PHP

1. Tổ chức mã nguồn

Hàm giúp phân chia mã nguồn thành các khối chức năng rõ ràng, dễ quản lý và bảo trì.

2. Tái sử dụng code

Viết một lần, sử dụng nhiều lần, giúp tiết kiệm thời gian và công sức trong quá trình phát triển.

3. Xây dựng ứng dụng hiệu quả

Tối ưu hóa hiệu suất và bộ nhớ thông qua việc tổ chức mã nguồn thành các hàm chuyên biệt.

Nội Dung Bài Giảng

Tổng quan về hàm trong PHP

Khái niệm, vai trò và lợi ích của hàm trong lập trình PHP

Khai báo hàm

Cú pháp, tham số, giá trị trả về và phạm vi của hàm

Sử dụng hàm

Cách gọi hàm, truyền tham số và xử lý giá trị trả về

Thư viện hàm

Các hàm có sẵn trong PHP và cách tổ chức hàm thành thư viện

Bài tập thực hành

Áp dụng kiến thức đã học vào các bài tập cụ thể

1. Tổng quan về hàm trong PHP

Khái niệm hàm

Hàm trong PHP là một khối mã nguồn thực hiện một tác vụ cụ thể và có thể được gọi đi gọi lại nhiều lần.

Tái sử dụng mã nguồn

Khi sử dụng hàm, chúng ta đang áp dụng nguyên tắc "viết một lần, sử dụng nhiều lần" - một trong những nguyên tắc quan trọng nhất trong lập trình.

Dễ quản lý mã nguồn

Hàm giúp chia nhỏ chương trình thành các phần có chức năng riêng biệt, dễ quản lý và bảo trì.

Lợi ích của việc sử dụng hàm

1. Tái sử dụng mã nguồn

Viết mã một lần và sử dụng lại nhiều lần, giúp tiết kiệm thời gian và công sức.

2. Tổ chức mã nguồn

Phân chia chương trình thành các đơn vị nhỏ hơn, dễ đọc và dễ bảo trì.

3. Bảo mật dữ liệu

Hạn chế phạm vi truy cập của biến, bảo vệ dữ liệu khỏi sự thay đổi ngoài ý muốn.

Đối với dự án lớn, việc sử dụng hàm còn giúp các thành viên trong nhóm làm việc độc lập trên các chức năng khác nhau mà không ảnh hưởng đến nhau.

2.1. Phân loại hàm trong PHP

2.1.1. Hàm do người dùng định nghĩa

Là những hàm do lập trình viên tự tạo ra để giải quyết các vấn đề cụ thể trong ứng dụng của mình.

```
function xinChao() {
    echo "Xin chào các bạn!";
}
```

2.1.2. Hàm có sẵn trong PHP

Là những hàm được tích hợp sẵn trong PHP, giúp thực hiện các tác vụ phổ biến như xử lý chuỗi, mảng, thời gian...

```
$chuoi = "Học PHP";
echo strlen($chuoi); // Hiển thị: 7
```

3.1. Khai báo hàm trong PHP

Để khai báo một hàm trong PHP, chúng ta sử dụng từ khóa **function**, theo sau là tên hàm và một cặp ngoặc đơn. Nội dung của hàm được đặt trong cặp dấu ngoặc nhọn.

Cú pháp cơ bản

```
function tenHam() {
    // Các câu lệnh thực thi
    echo "Đây là nội dung của hàm";
}
```

Quy tắc đặt tên

Tên hàm phải bắt đầu bằng chữ cái hoặc dấu gạch dưới, không được bắt đầu bằng số.

Quy ước đặt tên

Tên hàm không phân biệt hoa thường trong PHP, nhưng theo quy ước, chúng ta nên sử dụng kiểu camelCase (từ đầu viết thường, các từ sau viết hoa chữ cái đầu).

Tham số trong hàm

1. Định nghĩa tham số

Tham số là những giá trị được truyền vào hàm khi gọi hàm. Chúng ta có thể định nghĩa nhiều tham số cho một hàm, được phân cách bằng dấu phẩy.

2. Sử dụng tham số

```
function tinhTong($a, $b) {
    $tong = $a + $b;
    echo "Tổng của $a và $b là: $tong";
}
```

Trong ví dụ này, $a và $b là hai tham số của hàm tinhTong().

3. Kiểu dữ liệu cho tham số

Khi khai báo tham số, chúng ta có thể quy định kiểu dữ liệu cho tham số (từ PHP 7.0) để đảm bảo rằng dữ liệu đầu vào đúng với yêu cầu của hàm.

Tham số mặc định

PHP cho phép chúng ta thiết lập giá trị mặc định cho tham số. Nếu người dùng không truyền giá trị cho tham số khi gọi hàm, giá trị mặc định sẽ được sử dụng.

1. Cú pháp

```
function chaoHoi($ten, $tuoi = 20) {
    echo "Xin chào $ten, bạn $tuoi tuổi.";
}
```

2. Đầy đủ tham số

```
// Gọi hàm với đầy đủ tham số
chaoHoi("Minh", 22);
// Kết quả: Xin chào Minh, bạn 22 tuổi.
```

3. Giá trị mặc định

```
// Gọi hàm với một tham số
chaoHoi("Lan");
// Kết quả: Xin chào Lan, bạn 20 tuổi.
```

Lưu ý rằng các tham số có giá trị mặc định phải được đặt sau các tham số không có giá trị mặc định.

Giá trị trả về của hàm

Hàm có thể trả về một giá trị bằng cách sử dụng từ khóa **return**. Giá trị trả về có thể là bất kỳ kiểu dữ liệu nào: số, chuỗi, mảng, đối tượng...

1. Cú pháp return

```
function tinhDienTich($chieuDai, $chieuRong) {
    $dienTich = $chieuDai * $chieuRong;
    return $dienTich;
}

$dientich = tinhDienTich(5, 3);
echo "Diện tích là: $dientich";  // Kết quả: Diện tích là: 15
```

2. Lưu ý quan trọng

Khi câu lệnh return được thực thi, hàm sẽ kết thúc ngay lập tức và trả về giá trị được chỉ định. Các câu lệnh sau return sẽ không được thực thi.

3. Các kiểu dữ liệu trả về

Hàm có thể trả về nhiều kiểu dữ liệu khác nhau như số, chuỗi, mảng, đối tượng, hoặc thậm chí là null.

Phạm vi biến trong hàm

1. Biến cục bộ (Local)

Được khai báo trong hàm và chỉ có thể truy cập trong hàm đó.

```
function test() {
    $x = 10;  // Biến cục bộ
    echo $x;
}
```

2. Biến toàn cục (Global)

Được khai báo bên ngoài hàm và có thể truy cập từ bất kỳ đâu trong script.

```
$y = 20;  // Biến toàn cục

function test2() {
    global $y; // Khai báo sử dụng biến toàn cục
    echo $y;
}
```

Để sử dụng biến toàn cục trong hàm, cần sử dụng từ khóa **global** hoặc mảng **$GLOBALS**. Điều này giúp PHP hiểu rằng chúng ta muốn truy cập biến bên ngoài hàm, không phải tạo một biến cục bộ mới.

2.2. Tham số truyền theo giá trị và tham chiếu

Truyền theo giá trị (mặc định)

Một bản sao của giá trị được truyền vào hàm. Thay đổi trong hàm không ảnh hưởng biến gốc.

```
function tang($x) {
    $x += 5;
}

$a = 10;
tang($a);
echo $a;  // Vẫn là 10
```

Truyền theo tham chiếu

Địa chỉ của biến được truyền vào hàm. Thay đổi trong hàm sẽ ảnh hưởng đến biến gốc.

```
function tang2(&$x) {
 $x += 5;
}

$a = 10;
tang2($a);
echo $a; // Giờ là 15
```

2.2. Sử dụng hàm trong PHP

Khai báo hàm

Sau khi khai báo hàm, chúng ta có thể gọi hàm bằng cách sử dụng tên hàm và truyền các tham số cần thiết (nếu có).

```
// Khai báo hàm
function tinhBinhPhuong($so) {
    return $so * $so;
}
```

Gọi hàm

Hàm có thể được gọi và lưu kết quả vào biến hoặc sử dụng trực tiếp trong biểu thức.

```
// Gọi hàm và lưu kết quả vào biến
$ketQua = tinhBinhPhuong(5);
echo "Bình phương của 5 là: $ketQua";  // Kết quả: Bình phương của 5 là: 25

// Gọi hàm trực tiếp trong biểu thức
echo "Bình phương của 4 là: " . tinhBinhPhuong(4);  // Kết quả: Bình phương của 4 là: 16
```

Lưu ý quan trọng

Lưu ý rằng hàm phải được khai báo trước khi được gọi, hoặc phải nằm trong một file được include trước khi gọi.

Ví dụ thực tế: Tính điểm trung bình

1. Khai báo hàm

```
// Hàm tính điểm trung bình của nhiều môn học
function tinhDiemTrungBinh() {
 // Lấy tất cả các tham số được truyền vào
 $danhSachDiem = func_get_args();
 $tongDiem = 0;
 $soMonHoc = count($danhSachDiem);

 // Tính tổng điểm
 foreach ($danhSachDiem as $diem) {
 $tongDiem += $diem;
 }

 // Tính và trả về điểm trung bình
 return $tongDiem / $soMonHoc;
}
```

2. Sử dụng hàm

```
// Sử dụng hàm
$diemTB = tinhDiemTrungBinh(8, 7.5, 9, 6.5);
echo "Điểm trung bình: " . number_format($diemTB, 1); // Kết quả: Điểm trung bình: 7.8
```

3. Giải thích

Hàm `tinhDiemTrungBinh()` nhận vào nhiều tham số và tính điểm trung bình. Sử dụng `func_get_args()` để lấy tất cả các tham số được truyền vào hàm.

1. Hàm đệ quy

Hàm đệ quy là hàm tự gọi lại chính nó. Đây là một kỹ thuật mạnh mẽ để giải quyết các bài toán có thể chia nhỏ thành các bài toán con tương tự.

2. Nguyên lý hoạt động

Mỗi lần gọi đệ quy, bài toán được giảm kích thước cho đến khi đạt đến trường hợp cơ sở ($n <= 1).

3. Lưu ý quan trọng

Khi sử dụng đệ quy, cần đảm bảo rằng hàm sẽ kết thúc sau một số hữu hạn bước để tránh tràn ngăn xếp (stack overflow).

4. Ví dụ: Tính giai thừa

```
function giaiThua($n) {
    if ($n <= 1) {
        return 1;
    } else {
        return $n * giaiThua($n - 1);
    }
}

echo "5! = " . giaiThua(5);
// Kết quả: 5! = 120
```

Hàm ẩn danh (Anonymous Function)

Hàm ẩn danh là hàm không có tên và thường được sử dụng như tham số cho các hàm khác hoặc được gán cho biến.

```
// Gán hàm ẩn danh cho biến
$chao = function($ten) {
    return "Xin chào $ten!";
};

// Gọi hàm thông qua biến
echo $chao("Hùng");  // Kết quả: Xin chào Hùng!

// Sử dụng hàm ẩn danh làm callback
$mang = [1, 2, 3, 4, 5];
$mangBinhPhuong = array_map(function($so) {
    return $so * $so;
}, $mang);

// $mangBinhPhuong bây giờ là [1, 4, 9, 16, 25]
```

1. Không tên

Hàm không có tên cụ thể khi khai báo

2. Gán cho biến

Có thể gán cho biến để tái sử dụng

3. Arrow function

Cú pháp rút gọn từ PHP 7.4: fn($x) => $x * 2

2.3. Thư viện hàm trong PHP

PHP có một thư viện phong phú các hàm có sẵn để thực hiện nhiều tác vụ phổ biến. Dưới đây là một số nhóm hàm quan trọng:

Xử lý chuỗi

strlen(), strpos(), substr(), str_replace(), explode(), implode()...

Xử lý mảng

count(), array_push(), array_pop(), array_merge(), sort(), array_filter()...

Làm việc với CSDL

mysqli_connect(), mysqli_query(), PDO::prepare(), PDO::execute()...

Để sử dụng hiệu quả PHP, bạn nên làm quen với các hàm có sẵn này để tránh việc "tự phát minh lại bánh xe" - viết lại những hàm đã có sẵn trong ngôn ngữ.

Các hàm xử lý chuỗi phổ biến

1 strlen()

Đếm số ký tự trong chuỗi, bao gồm cả khoảng trắng.

```
$chuoi = "Học PHP";
echo strlen($chuoi); // Kết quả: 7

$chuoiTiengViet = "Xin chào";
echo strlen($chuoiTiengViet); // Kết quả: 8 (tính cả dấu)
```

2 strpos()

Tìm vị trí xuất hiện đầu tiên của chuỗi con trong chuỗi. Trả về FALSE nếu không tìm thấy.

```
$chuoi = "Học PHP cơ bản và PHP nâng cao";
$viTri = strpos($chuoi, "PHP");
echo $viTri; // Kết quả: 4

// Kiểm tra tồn tại
if (strpos($chuoi, "Java") === false) {
    echo "Không tìm thấy Java";
}
```

3 str_replace()

Thay thế tất cả các lần xuất hiện của chuỗi tìm kiếm bằng chuỗi thay thế.

```
$chuoi = "Học PHP cơ bản và PHP nâng cao";
$ketQua = str_replace("PHP", "Laravel", $chuoi);
echo $ketQua; // Kết quả: "Học Laravel cơ bản và Laravel nâng cao"

// Thay thế nhiều chuỗi cùng lúc
$ketQua2 = str_replace(
    ["PHP", "cơ bản"],
    ["JavaScript", "căn bản"],
    $chuoi
);
echo $ketQua2; // "Học JavaScript căn bản và JavaScript nâng cao"
```

4 substr()

Lấy một phần của chuỗi dựa trên vị trí bắt đầu và độ dài.

```
$chuoi = "Học lập trình PHP";
echo substr($chuoi, 0, 3); // Kết quả: "Học"
echo substr($chuoi, 4, 9); // Kết quả: "lập trình"
echo substr($chuoi, -3); // Kết quả: "PHP" (3 ký tự từ cuối)
echo substr($chuoi, 4); // Kết quả: "lập trình PHP" (từ vị trí 4 đến hết)
```

5 strtolower() / strtoupper()

Chuyển đổi chuỗi thành chữ thường hoặc chữ hoa.

```
$chuoi = "Học PHP";
echo strtolower($chuoi); // Kết quả: "học php"
echo strtoupper($chuoi); // Kết quả: "HỌC PHP"

// Lưu ý: Không hoạt động tốt với tiếng Việt có dấu
$tiengViet = "Tiếng Việt";
echo strtolower($tiengViet); // Không đổi được dấu
// Sử dụng mb_strtolower() cho tiếng Việt có dấu
```

Các hàm xử lý mảng phổ biến

1. Tạo và Thêm phần tử

```
// Tạo mảng
$fruits = ["táo", "cam", "chuối"];

// Thêm phần tử vào mảng
array_push($fruits, "xoài");
// $fruits bây giờ là ["táo", "cam", "chuối", "xoài"]
```

2. Đếm và Sắp xếp

```
// Đếm số phần tử
echo count($fruits);  // Kết quả: 4

// Sắp xếp mảng
sort($fruits);
// $fruits bây giờ là ["cam", "chuối", "táo", "xoài"]
```

3. Lọc mảng

```
// Lọc mảng
$numbers = [1, 2, 3, 4, 5, 6];
$evenNumbers = array_filter($numbers, function($n) {
 return $n % 2 == 0;
});
// $evenNumbers là [2, 4, 6]
```

4. Áp dụng hàm

```
// Áp dụng hàm cho mỗi phần tử
$doubled = array_map(function($n) {
 return $n * 2;
}, $numbers);
// $doubled là [2, 4, 6, 8, 10, 12]
```

1. Tổ chức mã nguồn thành thư viện

Khi dự án PHP lớn dần, việc tổ chức mã nguồn thành các file thư viện riêng biệt là rất quan trọng để dễ bảo trì và mở rộng.

1.1. Tạo file thư viện

```
// functions.php
function kiemTraEmail($email) {
    return filter_var($email,
        FILTER_VALIDATE_EMAIL);
}

function maHoaMatKhau($password) {
    return password_hash($password,
        PASSWORD_DEFAULT);
}

// Nhiều hàm khác...
```

1.2. Sử dụng trong file khác

```
// index.php
require_once 'functions.php';

$email = "user@example.com";
if (kiemTraEmail($email)) {
    echo "Email hợp lệ!";
} else {
    echo "Email không hợp lệ!";
}

$password = "123456";
$hashedPassword = maHoaMatKhau($password);
echo $hashedPassword;
```

Sử dụng **require_once** hoặc **include_once** để đảm bảo file thư viện chỉ được nạp một lần.

Viết tài liệu cho hàm

1. Mục đích

Viết tài liệu cho hàm là một thực hành tốt giúp người đọc code hiểu được mục đích, cách sử dụng và hành vi của hàm mà không cần đọc toàn bộ mã nguồn.

2. Ví dụ PHPDoc

```
/**
 * Tính tổng của hai số
 *
 * @param int $a Số thứ nhất
 * @param int $b Số thứ hai
 * @return int Tổng của hai số
 * @throws InvalidArgumentException Nếu tham số không phải là số
 */
function tinhTong($a, $b) {
    if (!is_numeric($a) || !is_numeric($b)) {
        throw new InvalidArgumentException("Tham số phải là số");
    }
    return $a + $b;
}
```

3. Lợi ích

Sử dụng phong cách PHPDoc với các thẻ @param, @return, @throws... giúp các IDE hiện đại như PHPStorm, VS Code cung cấp gợi ý và kiểm tra lỗi tốt hơn.

3. Bài tập thực hành

Bài tập 1: Viết hàm kiểm tra số nguyên tố

```
/**
 * Kiểm tra xem một số có phải là số nguyên tố hay không
 *
 * @param int $n Số cần kiểm tra
 * @return bool True nếu là số nguyên tố, False nếu không phải
 */
function laSoNguyenTo($n) {
    // Viết mã của bạn ở đây
    // Gợi ý: Số nguyên tố là số chỉ chia hết cho 1 và chính nó
}
```

Hãy thử hoàn thiện hàm này và kiểm tra với các số như 2, 3, 4, 7, 9, 11.

Bài tập 2: Tạo thư viện xử lý chuỗi

Hãy tạo một file **string_utils.php** chứa các hàm xử lý chuỗi sau:

1. Đảo ngược chuỗi

```
function daoNguocChuoi($str) {
    // Viết mã của bạn ở đây
    // Gợi ý: Sử dụng hàm strrev()
    // hoặc viết thuật toán đảo ngược
}
```

2. Đếm số từ trong chuỗi

```
function demSoTu($str) {
    // Viết mã của bạn ở đây
    // Gợi ý: Sử dụng hàm str_word_count()
}
```

3. Viết hoa chữ cái đầu mỗi từ

```
function vietHoaChuCaiDau($str) {
    // Viết mã của bạn ở đây
    // Gợi ý: Sử dụng hàm ucwords()
}
```

4. Mã hóa chuỗi đơn giản

```
function maHoaChuoi($str, $key) {
    // Viết mã của bạn ở đây
    // Gợi ý: Dịch chuyển mỗi ký tự
    // theo giá trị của $key
}
```

Sau đó tạo file **test.php** để kiểm tra các hàm này với nhiều chuỗi khác nhau.

Bài tập 3: Quản lý sinh viên

Tạo một hệ thống quản lý sinh viên đơn giản sử dụng hàm. Hệ thống bao gồm các chức năng:

Tạo các hàm thêm sinh viên

Hàm nhận thông tin cơ bản của sinh viên như tên, tuổi, mã sinh viên và lưu vào mảng.

Tạo hàm hiển thị danh sách sinh viên

Hàm hiển thị toàn bộ thông tin sinh viên đã lưu trữ theo định dạng bảng.

Tạo hàm tìm kiếm sinh viên

Hàm cho phép tìm kiếm sinh viên theo tên hoặc mã sinh viên.

Tạo hàm tính điểm trung bình

Hàm tính điểm trung bình cho mỗi sinh viên dựa trên điểm các môn học.

Tổ chức mã nguồn thành các file riêng biệt: **functions.php** (chứa các hàm), **data.php** (chứa dữ liệu), và **index.php** (file chính để chạy ứng dụng).

Hướng dẫn thực hiện:

1. Cấu trúc dữ liệu sinh viên

```
// Mỗi sinh viên sẽ được lưu dưới dạng mảng kết hợp:
$sinhVien = [
    'maSV' => 'SV001',
    'hoTen' => 'Nguyễn Văn A',
    'tuoi' => 20,
    'diemMonHoc' => [
        'Toán' => 8.5,
        'Lý' => 7.0,
        'Hóa' => 9.0
    ]
];
```

2. File functions.php

Tạo các hàm sau:

```
/**
 * Thêm sinh viên mới vào hệ thống
 * @param array &$danhSachSV Mảng chứa danh sách sinh viên
 * @param string $maSV Mã sinh viên
 * @param string $hoTen Họ tên sinh viên
 * @param int $tuoi Tuổi của sinh viên
 * @param array $diemMonHoc Mảng chứa điểm các môn học
 * @return bool True nếu thêm thành công, False nếu mã SV đã tồn tại
 */
function themSinhVien(&$danhSachSV, $maSV, $hoTen, $tuoi, $diemMonHoc = []) {
    // Kiểm tra mã sinh viên đã tồn tại chưa
    // Nếu chưa, thêm sinh viên mới vào mảng
}

/**
 * Hiển thị danh sách sinh viên dưới dạng bảng
 * @param array $danhSachSV Mảng chứa danh sách sinh viên
 */
function hienThiDanhSach($danhSachSV) {
    // Hiển thị tiêu đề bảng
    // Duyệt mảng và hiển thị thông tin từng sinh viên
}

/**
 * Tìm kiếm sinh viên theo tên hoặc mã SV
 * @param array $danhSachSV Mảng chứa danh sách sinh viên
 * @param string $tuKhoa Từ khóa tìm kiếm
 * @return array Mảng chứa các sinh viên tìm thấy
 */
function timKiemSinhVien($danhSachSV, $tuKhoa) {
    // Tạo mảng kết quả
    // Duyệt mảng và tìm sinh viên phù hợp
    // Trả về mảng kết quả
}

/**
 * Tính điểm trung bình của sinh viên
 * @param array $diemMonHoc Mảng chứa điểm các môn học
 * @return float Điểm trung bình
 */
function tinhDiemTrungBinh($diemMonHoc) {
    // Tính tổng điểm
    // Tính điểm trung bình và trả về kết quả
}
```

3. File data.php

```
$danhSachSinhVien = [];

// Thêm một số sinh viên mẫu
themSinhVien($danhSachSinhVien, 'SV001', 'Nguyễn Văn A', 20, [
    'Toán' => 8.5, 'Lý' => 7.0, 'Hóa' => 9.0
]);
themSinhVien($danhSachSinhVien, 'SV002', 'Trần Thị B', 19, [
    'Toán' => 9.0, 'Lý' => 8.5, 'Hóa' => 7.5
]);
```

4. File index.php

```
// Nhúng các file cần thiết
require_once 'functions.php';
require_once 'data.php';

// Hiển thị menu chức năng
echo "QUẢN LÝ SINH VIÊN\n";
echo "1. Xem danh sách sinh viên\n";
echo "2. Thêm sinh viên mới\n";
echo "3. Tìm kiếm sinh viên\n";
echo "4. Thoát\n";

// Xử lý lựa chọn của người dùng
// ...

// Gọi các hàm tương ứng với lựa chọn
```

Yêu cầu bổ sung:

1. Thêm chức năng sắp xếp sinh viên theo điểm trung bình
2. Thêm chức năng xóa sinh viên theo mã SV
3. Thêm chức năng cập nhật thông tin sinh viên

Hướng dẫn giải yêu cầu bổ sung:

1. Chức năng sắp xếp sinh viên theo điểm trung bình

```
/**
 * Sắp xếp danh sách sinh viên theo điểm trung bình
 * @param array &$danhSachSV Mảng chứa danh sách sinh viên
 * @param string $huong Hướng sắp xếp ('tang' hoặc 'giam')
 */
function sapXepTheoDiemTB(&$danhSachSV, $huong = 'giam') {
    // Tính điểm trung bình cho mỗi sinh viên và lưu tạm
    foreach ($danhSachSV as &$sv) {
        $sv['diemTB'] = tinhDiemTrungBinh($sv['diemMonHoc']);
    }

    // Sắp xếp mảng
    usort($danhSachSV, function($a, $b) use ($huong) {
        if ($huong == 'tang') {
            return $a['diemTB'] <=> $b['diemTB'];
        } else {
            return $b['diemTB'] <=> $a['diemTB'];
        }
    });

    // Xóa trường diemTB tạm
    foreach ($danhSachSV as &$sv) {
        unset($sv['diemTB']);
    }
}
```

2. Chức năng xóa sinh viên theo mã SV

```
/**
 * Xóa sinh viên khỏi danh sách
 * @param array &$danhSachSV Mảng chứa danh sách sinh viên
 * @param string $maSV Mã sinh viên cần xóa
 * @return bool True nếu xóa thành công, False nếu không tìm thấy
 */
function xoaSinhVien(&$danhSachSV, $maSV) {
    foreach ($danhSachSV as $index => $sv) {
        if ($sv['maSV'] == $maSV) {
            // Xóa sinh viên khỏi mảng
            array_splice($danhSachSV, $index, 1);
            return true;
        }
    }
    return false; // Không tìm thấy sinh viên
}
```

3. Chức năng cập nhật thông tin sinh viên

```
/**
 * Cập nhật thông tin sinh viên
 * @param array &$danhSachSV Mảng chứa danh sách sinh viên
 * @param string $maSV Mã sinh viên cần cập nhật
 * @param string $hoTen Họ tên sinh viên (để null nếu không cập nhật)
 * @param int $tuoi Tuổi của sinh viên (để null nếu không cập nhật)
 * @param array $diemMonHoc Mảng chứa điểm các môn học (để null nếu không cập nhật)
 * @return bool True nếu cập nhật thành công, False nếu không tìm thấy
 */
function capNhatSinhVien(&$danhSachSV, $maSV, $hoTen = null, $tuoi = null, $diemMonHoc = null) {
    foreach ($danhSachSV as &$sv) {
        if ($sv['maSV'] == $maSV) {
            // Cập nhật các trường nếu được cung cấp
            if ($hoTen !== null) $sv['hoTen'] = $hoTen;
            if ($tuoi !== null) $sv['tuoi'] = $tuoi;
            if ($diemMonHoc !== null) $sv['diemMonHoc'] = $diemMonHoc;
            return true;
        }
    }
    return false; // Không tìm thấy sinh viên
}
```