

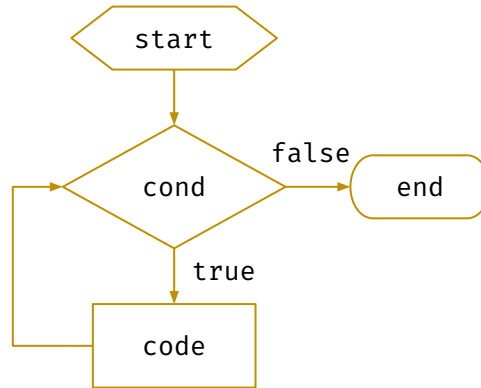
While, do while loops

Thực hiện một hành động (đoạn code) lặp đi lặp lại cho đến khi điều kiện trở thành **false**

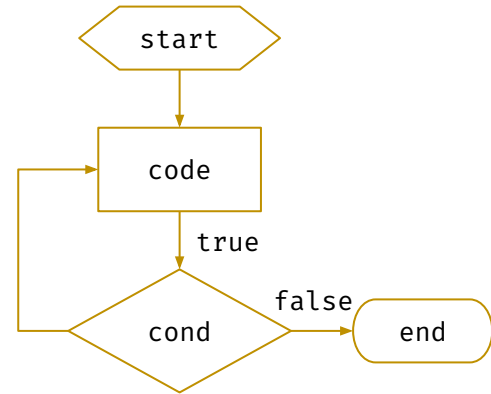
Cú pháp:

```
while (condition) {  
    // code  
}
```

```
do {  
    // code  
} while (condition);
```



While loop



Do while loop

While, do while loops

Ví dụ:

```
let sum = 0;  
let n = 0;
```

```
while (n < 10) {  
    sum += n;  
    n++;  
}
```

```
console.log(sum);
```

```
let sum = 0;  
let n = 0;
```

```
do {  
    sum += n;  
    n++;  
} while (n < 10);
```

```
console.log(sum);
```

```
let pass = "Người đàn ông quyền rũ  
nhất hành tinh 🥰";  
let answer = "";
```



```
while (answer !== pass) {  
    answer = prompt("Bạn có biết Ba  
là ai??");  
    if (answer !== pass) {  
        alert("Câu trả lời sai rồi!  
Lại nhé!");  
    }  
}
```

While, do while loops

Ví dụ:

```
let sum = 0;
let n = 0;

while (n < 10) {
    sum += n;
    n++;
}

console.log(sum);
```

Các bước thực hiện vòng lặp **while**

1. Kiểm tra $n < 10$
2. Tính $sum += n$
3. Tăng n ($n++$), lúc này $n = 1$
4. Quay trở về bước 2, kiểm tra $n < 10$
5. Lại tính $sum += n$
6. Lại tăng n ($n++$), lúc này $n = 2$
7. **Lặp lại** cho tới khi điều kiện $n < 10$ trở thành **false** (khi $n = 10$)
8. Kết thúc vòng lặp

Exercise

1. Viết chương trình cho phép nhập vào một loạt số, tính tổng tất cả các số đó, sử dụng **do while** hoặc **while**, chỉ dừng khi nhập số sai hoặc bấm cancel (có thể sử dụng phương thức `isNaN()` để kiểm tra số nhập vào có phải NaN hay không)

For loops

Vòng lặp **for** phức tạp hơn một chút so với **while**, và **do while**, tuy nhiên nó lại được sử dụng nhiều nhất

Cú pháp:

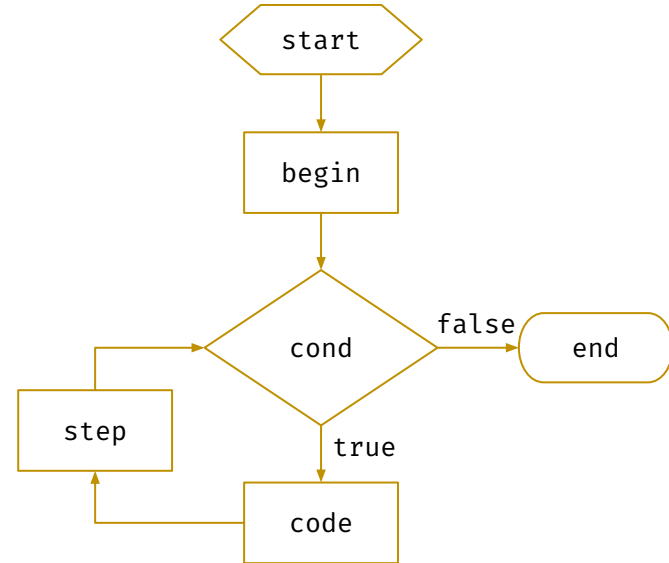
```
for (begin; condition; step) {  
    // code  
}
```



begin: khởi tạo (các) giá trị ban đầu

condition: (các) điều kiện của vòng lặp

step: cập nhật (các) giá trị sau mỗi vòng lặp



For loops

Ví dụ:

```
// tính tổng từ 1 đến 10
let sum = 0;

for (let i = 1; i ≤ 10; i++) {
    sum += i;
}

console.log(sum);
```

Các bước thực hiện vòng lặp **for**

1. Khởi tạo biến **i = 1** (chỉ khởi tạo 1 lần duy nhất)
2. Kiểm tra **i ≤ 10**
3. Tính **sum += i**
4. Tăng **i (i++)**, lúc này **i = 2**
5. Quay trở về bước 2, kiểm tra **i ≤ 10**
6. Lại tính **sum += i**
7. Lại tăng **i (i++)**, lúc này **i = 3**
8. **Lặp lại** cho tới khi điều kiện **i ≤ 10** trở thành **false** (khi **i = 11**)
9. Kết thúc vòng lặp

For loops

Ví dụ:

```
// in ra i và j
for (let i = 0, j = 10; i < j; i++, j--) {
    console.log(` i = ${i}, j = ${j}` );
}
```

```
// i = 0, j = 10
// i = 1, j = 9
// i = 2, j = 8
// i = 3, j = 7
// i = 4, j = 6
```

Các bước thực hiện vòng lặp **for**

1. Khởi tạo biến **i = 0**, **j = 10**
2. Kiểm tra **i < j**
3. In ra **i, j**
4. Tăng **i** (**i++**), giảm **j** (**j--**), lúc này **i = 1**, **j = 9**
5. Quay trở về bước 2, kiểm tra **i < j**
6. **Lặp lại** cho tới khi điều kiện **i < j** thành **false** (khi **i = 5**, và **j = 5**)
7. Kết thúc vòng lặp

Exercise

1. Viết chương trình tính tổng từ $0 \rightarrow 1000$ sử dụng vòng lặp **for**
2. Viết chương trình tính tổng tất cả số lẻ trong khoảng $0 \rightarrow 1000$ sử dụng vòng lặp **for**
3. Viết chương trình tính tổng đồng thời in ra các số lẻ chia hết cho 3 trong khoảng $0 \rightarrow 1000$ sử dụng vòng lặp **for**
4. Viết chương trình tính tổng đồng thời in ra các số chia hết cho cả 3, 5 và 7 nằm trong khoảng $0 \rightarrow 1000$ sử dụng vòng lặp **for**
5. Viết chương trình in ra bình phương đồng thời tính tổng bình phương các số chia hết cho 3 hoặc 5 hoặc 7 trong khoảng $0 \rightarrow 100$

Break, continue

Câu lệnh **break** **ngay lập tức** ngắt vòng lặp hiện tại, các câu lệnh bên dưới bị bỏ qua

```
let i = 1;
while (true) {
  let age = +prompt("Bạn nhiêu tuổi?");

  if (!isNaN(age) && age > 18) {
    break;
  }

  console.log(i++);
}
```

Câu lệnh **continue** dừng lần lặp hiện tại, chuyển lớp lần lặp tiếp theo, các câu lệnh bên dưới cũng bị bỏ qua

```
for (let i = 0; i < 10; i++) {
  if (i % 2 == 1) {
    continue;
  }

  console.log(i);
}
```

Inner loops, label

Các vòng lặp (hay các cấu trúc điều khiển khác) có thể sử dụng lồng nhau để giải quyết những bài toán phức tạp hơn, đồng thời cũng có thể đặt “nhãn” (label) cho vòng lặp để sử dụng với câu lệnh **break**,

```
for (let i = 0; i < 5; i++) {  
  continue  
  for (let j = i + 1; j < 5; j++) {  
    console.log(`i = ${i}, j = ${j}`);  
  }  
}
```



Inner loops, label

Các vòng lặp (hay các cấu trúc điều khiển khác) có thể sử dụng lồng nhau để giải quyết những bài toán phức tạp hơn, đồng thời cũng có thể đặt “nhãn” (label) cho vòng lặp để sử dụng với câu lệnh **break**,

```
outerLoop: for (let i = 0; i < 5; i++) {  
  continue  
  innerLoop: for (let j = i + 1; j < 5; j++) {  
    if (i > 3) {  
      break outerLoop;  
    }  
  
    console.log(`i = ${i}, j = ${j}`);  
  }  
}
```

Loops

Một số lưu ý khi sử dụng các vòng lặp

1. Cần chú ý điều kiện dừng, tránh vòng lặp vô hạn gây “đơ” máy tính



2. Vòng lặp **for** có thể bỏ qua một, hoặc tất cả biểu thức bên trong (), ví dụ:

```
let i = 0;  
for (; i < 10; i++) {  
  // code  
}
```

```
let i = 0;  
for (;;) {  
  // code, condition, step  
}
```

3. **break**, **continue** chỉ hoạt động với **if else**, không hoạt động với

Debugs

Ba Nguyễn

Homework

1. VCT tính và in bảng cửu chương, sử dụng vòng lặp for lồng nhau
2. VCT in ra màn hình nếu số chia hết cho 3 thì in “Fizz”, chia hết cho 5 thì in “Buzz”, chia hết cho cả 3 và 5 thì in “FizzBuzz”, không chia hết cho cả 3 và 5 thì in “BizzFuzz”. Số trong khoảng $0 \rightarrow 100$
3. VCT tính và in ra tổng bội chung của 3 và 5 trong khoảng $0 \rightarrow 1000$
4. VCT kiểm tra và in ra một số có phải số nguyên tố hay không
5. VCT kiểm tra và in ra các số nguyên tố trong khoảng $0 \rightarrow 1000$
6. VCT nhập vào 2 số a, b kiểm tra và in ra các số nguyên tố trong khoảng $a \rightarrow b$
7. VCT in ra bảng cửu chương ngược (từ $10 \rightarrow 1$)
8. VCT in ra chữ số đầu và cuối của một số. VD $12345 \rightarrow 15$

Homework

1. Viết chương trình in dãy số Fibonacci
2. Viết chương trình tìm bội chung nhỏ nhất, ước chung lớn nhất của 2 số
3. Viết chương trình in ra các pattern sau



```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

Homework

1. Viết chương trình in ra các pattern sau



1	1	1	1	1	1	5 5 5 5 5
2 2	2 2	2 2	2 5	2 3	2 1 2	4 4 4 4
3 3 3	3 3 3	3 3 3	3 6 8	4 5 6	3 2 1 2 3	3 3 3
4 4 4 4	4 4 4 4	4 4 4 4	4 7 9 10	7 8 9 10	2 1 2	2 2
5 5 5 5 5	5 5 5 5 5	5 5 5 5 5			1	1

1	1	1 2 3	1	1 2 3	3 2 1	1 2 3 4 5
1 2	2 1		1 1	1 2	2 1	2 3 4 5 1
1 2 3	3 2 1	3	1 2 1	1	1	3 4 5 1 2
1 2	4 3 2 1	2 3	1 3 3 1	1 2	2 1	4 5 1 2 3
1	5 4 3 2 1	1 2 3	1 4 6 4 1	1 2 3	3 2 1	5 1 2 3 4

Functions

Ba Nguyễn

Functions

Functions (hàm - phương thức) được sử dụng để “đóng gói” các “khối mã” (**code block**), cho phép tái sử dụng code, giảm thiểu code, giảm lỗi,

.... 🤔

Cú pháp khai báo:

💡 Tên hàm cũng sử dụng phong cách **camelCase**

// function declaration

```
function funcName() {  
    // code  
}
```

// run

```
funcName();
```

Biểu thức:

// function expression

```
let funcName = function () {  
    // code  
}
```

// run

```
funcName();
```

Functions

Ví dụ:

```
function isOdd() {  
    let n = +prompt("Nhập một số");  
    if (isNaN(n)) {  
        console.log("Vui lòng nhập một số hợp lệ!");  
    } else {  
        if (n % 2 == 1) {  
            console.log(n + " là số lẻ!");  
        } else {  
            console.log(n + " không phải là số lẻ!");  
        }  
    }  
}
```

```
// call function  
isOdd();  
isOdd();  
isOdd();  
isOdd();
```



Exercise

1. Viết hàm **maxOfThree()** cho phép nhập ba số, in ra số lớn nhất
2. Viết hàm **max()** cho phép nhập lượng số tùy ý in ra số lớn nhất
3. Viết hàm **isPrime()** cho phép nhập một số, kiểm tra và in ra số đó có phải số nguyên tố hay không

Local vs Global variables

Một biến được khai báo bên trong hàm **chỉ tồn tại bên trong hàm** đó, biến đó được gọi là **local variable**

```
function local() {  
    let name = "Ba";  
    console.log(name);  
}  
local(); // "Ba"  
console.log(name); // ❌ error
```

Một biến được khai báo bên ngoài tất cả các hàm (khối code) được gọi là **global variable**, hàm có thể truy cập tới biến khai báo bên ngoài

```
let name = "Ba";  
function global() {  
    console.log(name);  
}  
global(); // "Ba"  
console.log(name); // "Ba"
```


💡 Hàm chỉ tham chiếu tới biến bên ngoài, **nếu** trong hàm không có biến **local** nào trùng tên ngoài ra, **không** nên tham chiếu trực tiếp tới

Function Parameters

Tham số (đối số) cho phép truyền giá trị từ bên ngoài vào hàm, tham số được khai báo cùng với hàm. Khi một hàm được gọi, nó sẽ **sao chép** giá trị truyền vào và gán cho tham số tương ứng

```
function isOdd(number) {  
    // khai báo hàm với tham số number  
    if (number % 2 === 0) {  
        console.log(number + " không phải số lẻ");  
    } else {  
        console.log(number + " là số lẻ");  
    }  
}  
  
// gọi hàm  
isOdd(1); // number = 1
```

```
let outName = "Ba";  
  
function demo(inName) {  
    inName = "Béo Ú";  
    console.log(inName);  
}  
  
demo(outName); // "Béo Ú"  
console.log(outName); // "Ba"
```



Default Parameters

Khi hàm được gọi mà không truyền giá trị, tham số sẽ nhận giá trị **undefined**. Có chỉ định một giá trị mặc định cho tham số sẽ được sử dụng trong trường hợp đó:

```
function hi(name) {  
  console.log("Hello " + name);  
}
```

```
hi(); // "Hello undefined"  
hi("Ba"); // "Hello Ba"
```

```
function hi(name = "stranger") {  
  console.log("Hello " + name);  
}
```

```
hi(); // "Hello stranger"  
hi("Ba"); // "Hello Ba"
```



Exercise

1. Viết hàm **printPrime(n)** in ra các số nguyên tố trong khoảng từ 0 → n, mặc định n = 100
2. Viết hàm **convertTemperature(temp, from, to)** chuyển đổi và in ra nhiệt độ từ Celsius sang Fahrenheit hoặc Kevin, mặc định sẽ chuyển từ Celsius sang Kevin