GENERALITAT VALENCIANA
Conselleria d'Educació, Investigació, Cultura i Esport

ies SERPIS | departament informàtica

Unidad 6 – Funciones tradicionales, anónimas y flecha

DWEC - 2º DAW

Beatriz Fuster Ochando

Unidad 2 – Índice

1. Funciones tradicionales

Supongamos que esta función tradicional devuelve la suma de dos parámetros que le pasemos:

```
function add (num1, num2) {
    return num1 + num2;
}
```

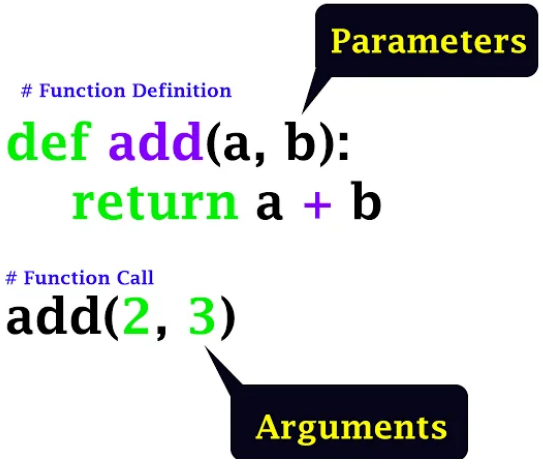
Cuando usemos esta función, simplemente la llamamos en el sitio deseado pasándole los argumentos apropiados. Por ejemplo:

```
console.log (add (3, 2)); // Se mostrará 5
```

Más información: https://www.youtube.com/watch?v=PEfw6xuj8Y0

1.1. Parámetros y argumentos

- Una función puede tener cero o más parámetros.
- Los parámetros son los nombres que aparecen en la definición de una función.
- Los argumentos son los valores que pasamos a (y que recibe) una función.



1.1. Parámetros y argumentos

Reglas de los parámetros:

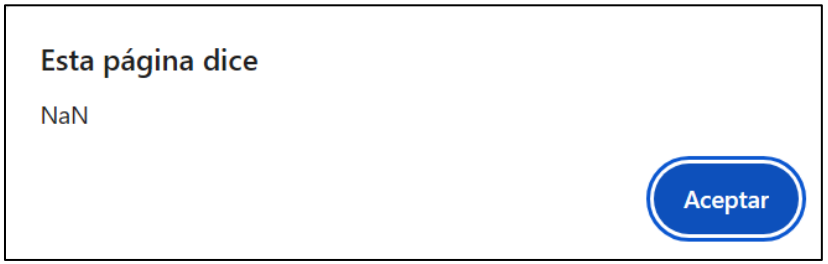
- No se especifica el tipo de los parámetros.
- No se verifican los tipos de los argumentos.
- No se comprueba el número de los argumentos recibidos.
  - Argumentos por defecto
  - Argumentos por exceso

1.1. Parámetros y argumentos

Argumentos por defecto

Cuando llamamos a una función con menos argumentos de los declarados. Los valores que faltan no están definidos.

```
function suma(a, b) {
    return a + b;
}
var resultado = suma(4);
alert(resultado);
```

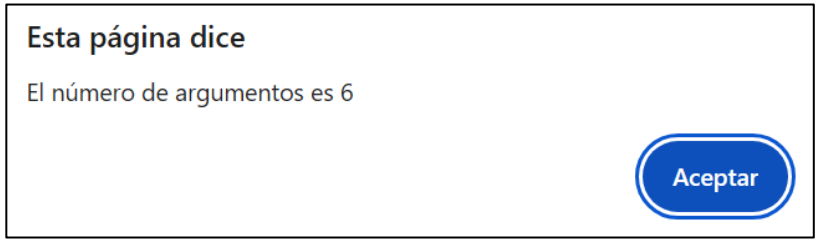


1.1. Parámetros y argumentos

Argumentos por exceso

Cuando llamamos a una función con más argumentos de los declarados. Los valores que nos llegan pueden capturarse a través de un objeto (incluido en la función) llamado arguments.

```
function valores() {
alert('El número de argumentos es ' + arguments.length);
}
valores(2, 5, 3, 2, 6, 7);
```



Más información: https://www.youtube.com/watch?v=5VVBrfWQ2Wk

2. Funciones anónimas

También podemos expresar la misma función como anónima. Son funciones que no tienen nombre, y por tanto se suelen asignar a una variable para que después puedan ser llamadas:

```
let addAnonymous = function (num1, num2) {
    return num1 + num2;
};
console.log (addAnonymous (3, 2));
```

2. Funciones anónimas: autoinvocadas

Son funciones anónimas que pueden ser llamadas a la vez que se declaran. Para llamarla utilizamos los 2 paréntesis al final y para indicar que todo es una expresión, ponemos toda la sentencia entre paréntesis.

```
(function () { alert (Hola);}());
```

Más información sobre funciones anónimas:
https://www.youtube.com/watch?v=GstPXAffmmI

2. Funciones flecha

5.2.4 FUNCIONES FLECHA

Hay otra manera de declarar funciones que se ha convertido en muy popular debido a su facilidad de escritura. Solo sirve para funciones anónimas y consiste en que no aparece la palabra **function** y en que una flecha separa los parámetros del cuerpo de la función. Ejemplo:

```
const triple=x=>3*x;
console.log(triple(20));
```

La definición de la función anónima es la sorprendente expresión:

```
x=>3*x
```

Que resulta ser equivalente a:

```
function(x){
  return 3*x;
}
```

2. Funciones flecha

El símbolo de la flecha separa los argumentos del cuerpo de la función, en el que, además, se sobrentiende la palabra **return**.

Evidentemente, es una notación para escribir más rápido. Si hay más de un parámetro, se deben colocar entre paréntesis. Ejemplo:

```
const media=(x,y)=>(x+y)/2;
console.log(media(10,20)); //Escribe 15
```

Esta función es un poco más compleja y requiere que los dos parámetros que utiliza la función estén entre paréntesis, si no la expresión fallaría. Por otro lado, si el cuerpo de la función es más complejo, requiere ser incluido entre llaves:

```
const sumatorio = (n)=>{
    let acu=0;
    for(let i=n;i>0;i--){
        acu+=i;
    }
    return acu;
}
console.log(sumatorio(3)); //Escribe 6, resultado de 3+2+1
```

2. Funciones flecha

Si en la función no hay parámetros, hay que colocar paréntesis vacíos en la posición que ocuparían los parámetros:

```
const hola = ()=>{
  console.log("Hola");
}
hola(); //Escribe Hola
```

Pero hay que tener en cuenta que, ante funciones complejas, las ventajas de las funciones flecha se diluyen:

```
const pares=(array)=>{
    let nPares=0;
    if(array instanceof Array){
        for(n of array){
            if(n%2===0){
                nPares++;
            }
        }
    }
    return nPares;
}
```

2. Funciones flecha

No parece un código que ahorre mucho respecto a forma clásica de escribir funciones anónimas:

```
const pares= function(array){
    let nPares=0;
    if(array instanceof Array){
        for(n of array){
            if(n%2===0){
                nPares++;
            }
        }
    }
    return nPares;
}
```

De ahí que lo habitual es que los programadores solo usen funciones flecha para definir funciones sencillas.