TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

FUNDAMENTOS DE PROGRAMACIÓN

UNA ESTRATEGIA CLAVE PARA ORGANIZAR EL CÓDIGO Y FOMENTAR LA MODULARIDAD

UNA ESTRATEGIA CLAVE PARA ORGANIZAR EL CÓDIGO Y FOMENTAR LA MODULARIDAD

El uso de funciones en programación permite estructurar tareas de forma clara y reutilizable. Esta práctica mejora la organización del código, facilita su mantenimiento y evita la repetición innecesaria de instrucciones. En este documento se exploran los distintos tipos de llamado de funciones, la manera en que pueden retornar valores y cómo su reutilización potencia la eficiencia del desarrollo de software.

Llamado y reutilización de funciones

Las funciones son bloques de código diseñados para realizar tareas específicas dentro de un programa. Su principal ventaja radica en la posibilidad de ser llamadas múltiples veces, evitando la repetición innecesaria de código y mejorando la organización del mismo (Moreno Pérez, 2015). Comprender cómo se realiza el llamado de funciones y su reutilización es fundamental para el desarrollo eficiente de software.

Llamado de funciones

El llamado de una función ocurre cuando se invoca su nombre dentro del código, seguido de paréntesis (). Dependiendo de la función, estos paréntesis pueden incluir argumentos que le proporcionan datos.

Llamado básico de una función

Cuando una función se define, esta no se ejecuta automáticamente. Para activarla, debe ser llamada explícitamente.

Figura 1. Llamado básico de función en Python

```python
def saludar():
    print("Hola, bienvenido a la programación!")

saludar()  # Llamado de la función
```

Figura 2. Llamado de función en Javascript

```javascript
function saludar() {
    console.log("Hola, bienvenido a la programación!");
}

saludar(); // Llamado de la función
```

En ambos ejemplos, la función saludar() es llamada sin argumentos y ejecuta la tarea definida en su interior.

Llamado con parámetros y retorno de valores

Algunas funciones requieren argumentos para procesar información y devolver un resultado.

Figura 3. Parámetros con retorno en Python

```python
def sumar(a, b):
    return a + b

resultado = sumar(5, 3)
print(resultado)  # Salida: 8
```

Figura 4. Parámetros con retorno en Javascript

```javascript
function sumar(a, b) {
    return a + b;
}

let resultado = sumar(5, 3);
console.log(resultado); // Salida: 8
```

Aquí, la función sumar() recibe dos argumentos (a y b), realiza una operación y devuelve el resultado.

Reutilización de funciones

Uno de los principales beneficios de las funciones es su capacidad de ser reutilizadas en distintos contextos dentro de un programa. Esto reduce la duplicación de código y mejora la modularidad del software (Moreno Pérez, 2015).

Reutilización de funciones en estructuras repetitivas

Las funciones pueden ser llamadas dentro de bucles para procesar múltiples valores.

Figura 5. Reutilización de funciones en Python

```python
def cuadrado(numero):
    return numero ** 2

for i in range(1, 6):
    print(f"El cuadrado de {i} es {cuadrado(i)}")
```

Figura 6. Reutilización de funciones en Javascript

```javascript
function cuadrado(numero) {
    return numero ** 2;
}

for (let i = 1; i <= 5; i++) {
    console.log(`El cuadrado de ${i} es ${cuadrado(i)}`);
}
```

Aquí, la función cuadrado() se reutiliza dentro de un bucle para calcular el cuadrado de varios números.

Funciones reutilizadas en múltiples partes del código

Cuando una función realiza una tarea común, puede ser utilizada en distintos lugares del código sin necesidad de repetir la lógica (Moreno Pérez, 2015).

Figura 7. Funciones reutilizadas en Python

```python
def convertir_mayusculas(texto):
    return texto.upper()

nombre = "ana"
ciudad = "madrid"

print(convertir_mayusculas(nombre))  # Salida: ANA
print(convertir_mayusculas(ciudad))  # Salida: MADRID
```

Figura 8. Funciones reutilizadas en Javascript

```javascript
function convertirMayusculas(texto) {
    return texto.toUpperCase();
}

let nombre = "ana";
let ciudad = "madrid";

console.log(convertirMayusculas(nombre)); // Salida: ANA
console.log(convertirMayusculas(ciudad)); // Salida: MADRID
```

En estos ejemplos, la función convertir_mayusculas() es utilizada en distintos puntos del código sin necesidad de repetir su implementación.

Funciones con retorno de valores

Las funciones con retorno de valores son aquellas que devuelven un resultado después de su ejecución. A diferencia de las funciones que solo realizan una acción (como imprimir un mensaje en pantalla), estas permiten capturar el valor producido para utilizarlo en otras partes del programa. Comprender su funcionamiento es fundamental para escribir código eficiente y modular (Menchaca García, 2010).

Concepto de retorno de valores en funciones

Cuando una función procesa datos y devuelve un resultado, este puede ser asignado a una variable, utilizado en cálculos o incluso pasado como argumento a otra función.

Para devolver un valor, la mayoría de los lenguajes utilizan la palabra clave return.

Retorno de un único valor

Figura 9. Retorno único en Python

```python
def cuadrado(numero):
    return numero ** 2  # Devuelve el cuadrado del número

resultado = cuadrado(4)
print(resultado)  # Salida: 16
```

Figura 10. Retiro único en Javascript

```javascript
function cuadrado(numero) {
    return numero ** 2;  // Devuelve el cuadrado del número
}

let resultado = cuadrado(4);
console.log(resultado); // Salida: 16
```

Aquí, la función cuadrado() recibe un número, calcula su cuadrado y devuelve el resultado con return. Luego, dicho resultado se almacena en una variable y se imprime en pantalla.

Importancia del uso de return

Una función sin return no devuelve ningún valor, lo que significa que su resultado no puede ser almacenado ni reutilizado.

Figura 11. Ejemplo de función sin retorno en Python

```python
def sin_retorno():
    print("Esta función solo imprime un mensaje.")

resultado = sin_retorno()
print(resultado)  # Salida: None
```

Figura 12. Ejemplo de función sin retorno en JavaScript

```javascript
function sinRetorno() {
    console.log("Esta función solo imprime un mensaje.");
}

let resultado = sinRetorno();
console.log(resultado); // Salida: undefined
```

En estos ejemplos, las funciones imprimen un mensaje, pero no devuelven un valor útil. Al intentar almacenar su resultado, se obtiene None en Python y undefined en JavaScript.

Retorno de múltiples valores

Algunos lenguajes permiten retornar más de un valor a la vez.

Figura 13. Retorno múltiple en Python (tuplas)

```python
def operaciones(a, b):
    suma = a + b
    resta = a - b
    return suma, resta  # Retorna dos valores

resultado_suma, resultado_resta = operaciones(10, 5)
print(resultado_suma)  # Salida: 15
print(resultado_resta)  # Salida: 5
```

Aquí, la función devuelve una tupla con dos valores, los cuales pueden ser desempaquetados en variables individuales.

Funciones con retorno condicional

Es posible definir funciones que devuelvan distintos valores según ciertas condiciones.

Figura 14. Retorno condicional en Python

```python
def evaluar_nota(nota):
    if nota >= 60:
        return "Aprobado"
    else:
        return "Reprobado"

print(evaluar_nota(75))  # Salida: Aprobado
print(evaluar_nota(50))  # Salida: Reprobado
```

Figura 15. Retorno condicional en Javascript

```javascript
function evaluarNota(nota) {
    return nota >= 60 ? "Aprobado" : "Reprobado";
}

console.log(evaluarNota(75)); // Salida: Aprobado
console.log(evaluarNota(50)); // Salida: Reprobado
```

Aquí, la función evalúa la nota y retorna diferentes valores dependiendo de la condición.

Buenas prácticas en el uso de funciones

El uso adecuado de funciones en la programación es esencial para garantizar un código limpio, modular y eficiente. Siguiendo principios de buenas prácticas, se pueden desarrollar aplicaciones más mantenibles, reutilizables y fáciles de depurar (Menchaca García, 2010). A continuación, se presentan las principales conclusiones sobre la correcta implementación de funciones.

- La modularidad mejora la legibilidad y mantenibilidad
- La claridad en los nombres de funciones es fundamental
- La correcta gestión de parámetros y argumentos evita errores
- El principio de responsabilidad única evita funciones monolíticas
- Evitar efectos secundarios innecesarios
- La documentación es clave para la comprensión del código
- Uso de funciones de orden superior para mayor flexibilidad

Bibliografía

- Menchaca García, F. R. (2010). Fundamentos de programación en Lenguaje C. Instituto Politécnico Nacional. https://elibro.net/es/lc/tecnologicadeloriente/titulos/74076
- Moreno Pérez, J. C. (2015). Programación. RA-MA Editorial. https://elibro.net/es/lc/tecnologicadeloriente/titulos/62476