# Conceptos y Paradigmas de Lenguajes de Programación

Trabajo integrador 2024


**Grupo N° 32 - Participantes:**

Dellarupe Franco - 21239/0
Ponce Gregorio - 21361/2
Romagnoli Leandro - 19505/9
Spadari Pedro - 21586/8


**Lenguajes asignados:**

Principal: JavaScript
Secundario: Python

**Bibliografía consultada:**


https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://tc39.es/ecma262/
https://docs.python.org/3.12/

**Punto A:**

**Abstracción:** capacidad de definir y usar estructuras u operaciones complejas sin necesidad de conocer todos sus detalles.

- JavaScript

```
1.    function calcularPromedio(numeros) {
2.        let suma = 0
3.        for (let i = 0; i < numeros.length; i++) {
4.            suma += numeros[i]
5.        }
6.        return suma / numeros.length
7.    }
8.    let listaNumeros = [10, 20, 30, 40, 50]
9.    let promedio = calcularPromedio(listaNumeros)
10.   console.log("El promedio es: ", promedio)
```

Este es un código en JavaScript el cual permite que una persona solo necesite saber qué parámetros debe enviar a la función para que esta calcule su promedio. No necesita saber cómo está implementada, solo utilizarla pasándole una lista de números.

- Python

```
1.    def calcular_promedio(numeros):
2.        suma = sum(numeros)
3.        return suma / len(numeros)
4.
5.    lista_numeros = [10, 20, 30, 40, 50]
6.    promedio = calcular_promedio(lista_numeros)
7.    print("El promedio es: ", promedio)
```

Este código hace lo mismo que el anterior, la única diferencia es la forma de declarar la función y que se utiliza una función ya integrada en el lenguaje, lo que demuestra también este principio, ya que no necesitamos saber cómo está definida la función para utilizarla y entender que suma todos los elementos de la lista de números pasada.

**Ortogonalidad:** un conjunto pequeño de constructores primitivos puede ser combinado a la hora de construir estructuras de control y datos.

- JavaScript

```
1.    let a = 5
2.    let b = 10
3.    let c = 15
4.
5.    if (a < b && b < c) {
6.        console.log("Los números están en orden
 ascendente.")
7.    } else {
```

```
8.        console.log("Los números no están en orden
 ascendente.")
9.    }
```

En este código se utilizan distintos operadores en una misma expresión booleana.

Se utiliza el operador lógico && que verifica que ambas sentencias sean verdaderas para dar paso a las instrucciones siguientes, pero dentro de cada uno de los lados del operador se utilizan operadores de comparación que sirven para verificar los valores.

Sigue el principio de ortogonalidad ya que, se pueden combinar los distintos constructores primitivos de manera de obtener expresiones complejas, sin perder la consistencia.

- Python

```
1.    a = 5
2.    b = 10
3.    c = 15
4.
5.    if a < b < c:
6.        print("Los números están en orden ascendente.")
7.    else:
8.        print("Los números no están en orden ascendente.")
```

Al igual que en el anterior, se utilizan dos tipos de operadores distintos para controlar el flujo del programa según el valor de las variables.

En este caso la instrucción a < b < c combina (a < b) and (b < c).

Sigue el principio de ortogonalidad porque construye expresiones complejas a partir de un conjunto de constructores primitivos de manera consistente.

**Punto B:**

**Estructura if**

- JavaScript

```
1.    let nota = 85
2.
3.    if (nota >= 90) {
4.        console.log("Aprobado - Excelente")
5.    } else if (nota >= 70) {
6.        console.log("Aprobado - Bien")
7.    } else {
8.        console.log("Desaprobado")
9.    }
```

- Python

```
1.    nota = 85
2.
3.    if nota >= 90:
```

```
4.          print("Aprobado - Excelente")
5.      elif nota >= 70:
6.          print("Aprobado - Bien")
7.      else:
8.          print("Desaprobado")
```

Las diferencias sintácticas con respecto a la estructura de control if son las siguientes:

1. En JavaScript los bloques de código que se ejecutan se encuentran encerrados entre llaves. En Python, luego del símbolo ":", es la indentación la que marca a qué instrucción corresponde cada bloque.
2. En JavaScript es obligatorio el uso de paréntesis para evaluar las condiciones. En Python si bien está permitido su uso, no es de carácter obligatorio.
3. En JavaScript para realizar un nuevo if en la sentencia else de un if previo, se debe usar 'else if', mientras que en Python se debe usar 'elif'.

**Estructura for**

- JavaScript

```
1.  let palabras = ["Hola", "Mundo", "JavaScript"];
2.
3.  for (let i = 0; i < palabras.length; i++) {
4.      if (palabras[i].length > 5) {
5.          console.log(palabras[i].toUpperCase());
6.      } else {
7.          console.log(palabras[i].toLowerCase());
8.      }
9.  }
```

- Python

```
1.  palabras = ["Hola", "Mundo", "Python"]
2.
3.  for palabra in palabras:
4.      print(palabra)
5.  else:
6.      print("Fin del loop!")
```

Las diferencias sintácticas con respecto a la estructura de control for son las siguientes:

1. En JavaScript se necesita inicializar una variable índice, marcar el límite y definir de qué forma se modifica en cada iteración el índice, todo esto está encerrado entre paréntesis y el bloque de código a reiterar se encierra entre llaves. Mientras que en el caso de Python se utiliza una variable temporal la cual será llamada para ser utilizada y que en cada repetición se modificará por el siguiente valor en la lista, el bloque de código a ejecutar se encuentra luego de ":" y tendrá que estar indentado.
2. En el caso de Python, se puede indicar un bloque else opcional, el cual se ejecutará cuando el bucle termine (excepto cuando termine debido a una sentencia break).

**Punto C:**

- JavaScript

```
1.    console.log(variable_no_declarada) // Error de
 semántica estática: error semántico estático al intentar
 acceder a una variable que nunca fue declarada.
2.
3.    let persona = {
4.        nombre: "Juan"
5.    };
6.
7.    console.log(persona.apellido.toUpperCase())// Error de
 semántica dinámica: error semántico dinámico al intentar
 acceder a la propiedad toUpperCase() de la propiedad
 apellido del objeto persona, la cual no está definida.
```

- Python

```
1.    def es_doble_digito(num):
2.        return (num > 9 & num < 100)
3.
4.    def resto(num1, num2):
5.        return num1 % num2
6.
7.    resultado1 = es_doble_digito() # Error de semántica
 estática: error semántico estático al llamar a la función
 es_doble_digito sin parámetros cuando tiene uno.
8.
9.    resultado2 = resto(10, 0)  # Error de semántica
 dinámica: error semántico dinámico al hacer módulo por 0.
10.   print(resultado1, resultado2)
```

**Punto D:**

Para este punto vamos a describir los tipos de variables permitidos tanto en JavaScript y Python según su tiempo de vida.

Comencemos definiendo que es tiempo de vida: El tiempo de vida de una variable o de un objeto es el periodo en el cual este se encuentra activo en memoria. Este periodo depende del tipo de objeto o de variable, cómo se crea y cómo es utilizado en el programa.

- JavaScript

```
1.    const variableGlobal1 = "Global solo lectura"
2.    var variableGlobal2= "Global lectura y escritura"
3.    function ejemplo(){
4.        var variableLocal= "Local a la función"
5.        if (true){
6.            let variableBloque= "Local al bloque if"
7.            console.log("Global 1: " + variableGlobal1)
8.            console.log("Global 2: " + variableGlobal2)
```

```
9.            console.log("Local 1: " + variableLocal)
10.           console.log("Local 2: " + variableBloque)
11.       }
12.   }
13.   ejemplo()
```

| Nombre | Momento alocación | R-valor | Alcance | Tiempo vida |
|---|---|---|---|---|
| variableGlobal1 | automático | "Global solo lectura" | 2-13 | 1-13 |
| variableGlobal2 | automático | indefinido (basura) | 1-13 | 1-13 |
| variableLocal | automático | indefinido (basura) | 3-12 | 3-12 |
| variableBloque | automático | "Local al bloque if" | 7-11 | 5-11 |

- Python

```
1.    variable_global = "Variable global"
2.
3.    def ejemplo():
4.        variable_local = "Variable local"
5.        print(variable_local)
6.        global variable_global # Se debe indicar con la
 palabra reservada global que se hace referencia a la
 variable previamente definida y no se quiere definir una
 nueva, local, con el mismo nombre
7.        variable_global += " (modificada dentro de
 ejemplo)"
8.
9.    print(variable_global)
10.   ejemplo()
11.   print(variable_global)
```

| Nombre | Momento alocación | R-valor | Alcance | Tiempo vida |
|---|---|---|---|---|
| variable_global | automático | "Variable global" | 2-11 | 1-11 |
| variable_local | automático | "Variable local" | 5-7 | 3-7 |