- A Cite un ejemplo de código para cada lenguaje seleccionado donde se aprecien
- B Ejemplifique la fortaleza del sistema de tipos de ambos lenguajes con al
- C Cite el paradigma de manejo de excepciones correspondiente a cada lenguaje
- D Conclusión sobre los lenguajes: Realice una entrevista a un usuario/s
- E Conclusión sobre el trabajo: Realice una conclusión de una carilla como

Ejercicio A)

Cite un ejemplo de código para cada lenguaje seleccionado donde se aprecien los distintos tipos de parámetros soportados y su uso. Cada fragmento puede tener una breve explicación textual. En la misma debe quedar claro el modo de ligadura que utiliza el lenguaje (posicional, por nombre, etc), si requieren o no ser tipados, o cualquier otra característica que considere relevante.

Python	JavaScript

`def greet(name, age=30):`
`  print(f"¡Hola {name}! Tienes {age} años")`
 	
`# Ejemplo de uso de la función 'greet'`
`greet("Juan")`
`greet("María", 25)`
 	
Esta función muestra un saludo personalizado con el nombre y la edad
 	
• **'name'** es obligatorio y se liga por posición.
• **'age'** es opcional y tiene un valor predeterminado de 30.
 	
En Python, los parámetros se ligan principalmente por posición, lo que significa que el orden de los argumentos en la llamada a la función es importante. Sin embargo, también es posible ligar parámetros por nombre al especificar el nombre del parámetro seguido de un signo igual y el valor correspondiente. Los parámetros pueden tener valores predeterminados, lo que los convierte en parámetros opcionales.
	`function calculateArea(width, height = 10) {`
	`  return width * height;`
	`}`
	
	`// Ejemplo de uso de la función 'calculateArea'`
	`console.log(calculateArea(5));`
	`console.log(calculateArea(3, 7));`
	
	Esta función calcula el área de un rectángulo dado su ancho y alto.
	
	• **'width'** es obligatorio y se liga por posición.
	• **'height'** es opcional y tiene un valor predeterminado de 10.
	
	En JavaScript, los parámetros se ligan por posición, al igual que en Python. Sin embargo, a diferencia de Python, no se pueden especificar valores predeterminados directamente en la declaración de la función. En su lugar, se puede asignar un valor predeterminado al parámetro dentro del cuerpo de la función utilizando la sintaxis = valor. Si no se proporciona un valor para un parámetro opcional, se utilizará su valor predeterminado.

Es importante tener en cuenta que tanto Python como JavaScript son lenguajes de tipado dinámico, lo que significa que no se requiere especificar explícitamente el tipo de los parámetros en la declaración de la función. Los parámetros pueden aceptar cualquier tipo de dato y su tipo puede cambiar durante la ejecución del programa.

Ejercicio B)

Ejemplifique la fortaleza del sistema de tipos de ambos lenguajes con al menos dos ejemplos de código (5 líneas mínimo, 10 líneas máximo) que sustenten la definición otorgada.

Python se considera un lenguaje de programación fuertemente tipado debido a la forma en que maneja los tipos de datos y las operaciones entre ellos. Hay varias razones por las cuales Python es fuertemente tipado:

- **`Seguridad y confiabilidad`** : Python garantiza que las operaciones se realicen solo entre tipos de datos compatibles. Esto evita comportamientos inesperados y errores difíciles de detectar en tiempo de ejecución.
- **`Detección temprana de errores`** : Python realiza una verificación de tipos en tiempo de ejecución, lo que significa que los errores relacionados con los tipos de datos se detectan antes de que el programa se ejecute por completo. Esto permite corregir los errores de forma rápida y eficiente durante la fase de desarrollo, en lugar de descubrirlos más tarde cuando pueden ser más difíciles de identificar y solucionar.
- **`Expresividad del código`** : Aunque Python no requiere la declaración explícita de tipos en las variables, los tipos de datos están presentes y las operaciones se evalúan en función de esos tipos. Esto permite una mayor expresividad del código al no tener que preocuparse por declarar tipos de variables innecesariamente. Sin embargo, es importante tener en cuenta que aunque las variables no tienen un tipo declarado, los valores que contienen sí tienen un tipo y deben ser coherentes con las operaciones que se realizan.
- **`Control de conversiones de tipos`** : Python requiere conversiones explícitas de tipos cuando es necesario realizar operaciones entre tipos incompatibles. Esto ayuda a evitar ambigüedades y errores sutiles al forzar al programador a indicar de manera explícita cómo se deben realizar las conversiones.

```
entero = 1
real = 9.0
print (entero + real)
cadena = “soy una cadena de texto”
print (entero + cadena)
```

En el ejemplo escrito anteriormente se demuestra la operación de suma entre tipos de datos compatibles (integer y double) y la operación entre tipos de datos no compatibles y por el cual se levanta una excepción (TypeError) al intentar realizar esta operación.

JavaScript se considera un lenguaje débilmente tipado debido a su enfoque en la flexibilidad y la capacidad de realizar conversiones automáticas de tipos de datos. Hay varias razones por las cuales JavaScript es débilmente tipado:

- **`Conversiones automáticas de tipos`** : JavaScript permite conversiones automáticas y coerciones de tipos en muchas operaciones. Esto significa que, en determinadas situaciones, JavaScript intentará convertir automáticamente un tipo de dato a otro para que la operación se pueda realizar.
- **`Tipado dinámico`** : En JavaScript, las variables no están asociadas a un tipo específico y pueden contener diferentes tipos de datos a lo largo del tiempo. Esto permite una mayor flexibilidad en el código, ya que una variable puede cambiar su tipo de dato sin requerir una declaración o asignación explícita de tipo.
- **`Operaciones sin restricciones de tipos`** : En JavaScript, se pueden realizar operaciones entre diferentes tipos de datos sin restricciones explícitas. Por ejemplo, se pueden sumar una cadena de texto y un número sin necesidad de realizar conversiones o declarar explícitamente el tipo de dato.

```
var entero = 1
var real = 9.0
console.log(entero + real);
var cadena = “soy una cadena de texto”
console.log(entero + cadena);
```

con este ejemplo que es el mismo que tenemos en python podemos denotar la diferencia entre un lenguaje fuerte y débilmente tipado, en este caso no se lanza ninguna excepción, sino que se printea correctamente 10 y luego se printea “1soy una cadena de texto”

`Ejercicio C)`

Cite el paradigma de manejo de excepciones correspondiente a cada lenguaje asignado, y coloque un ejemplo para cada uno que describa claramente el circuito posible al alcanzar una excepción y que considere todas las variantes de manejo.

JAVASCRIPT

Javascript utiliza manejo de excepciones por terminación , es decir, el controlador de excepciones realiza las acciones necesarias para manejar la excepción, pero no se retorna al punto donde se produjo la excepción (invocador), continúa su ejecución a partir de la finalización del manejador.

Posee 2 circuitos en los que clasificar su manejo:

Circuitos básicos de manejo de excepciones:

try-catch	try-finally
`try { // Código susceptible a lanzar una excepción throw new Error("¡Se ha producido un error!"); } catch (error) { // Manejo de la excepción console.log( "Se ha capturado una excepción: " + error.message ); }`	`try { // Código susceptible a lanzar una excepción throw new Error("¡Se ha producido un error!"); } finally { console.log("Este bloque siempre se ejecuta."); }` Bloque de código que siempre se ejecuta, independientemente de si se lanza una excepción o no

Circuitos avanzados de manejo de excepciones:

try-catch-finally	try-catch con múltiples catch
`try { // Código susceptible a lanzar una excepción throw new Error("¡Se ha producido un error!"); } catch (error) { // Manejo de la excepción console.log("Se ha capturado una excepción: " + error.message); } finally { console.log("Este bloque siempre se ejecuta."); }` Bloque de código que siempre se ejecuta, independientemente de si se lanza una excepción o no	`try { // Código susceptible a lanzar una excepción throw new TypeError("¡Se ha producido un error de tipo!"); } catch (error) { if (error instanceof ReferenceError) { console.log("Se ha capturado una excepción de referencia: " + err } else if (error instanceof TypeError) { console.log("Se ha capturado una excepción de tipo: " + error.mes } else { console.log("Se ha capturado una excepción: " + error.message); } }`

	Bloque de código que siempre se ejecuta, independientemente de si se lan excepción o no

¿Qué sucede si no se encuentra un manejador para la excepción ocurrida?

Tenemos 2 opciones:

- **`Plantear caso por defecto y manejar genéricamente`** Deberíamos plantear el caso en que no se haya encontrado el manejador específico y poner uno como default como el caso de arriba que pone un else si no cayó en ninguno de los errores y manejar el error de forma genérica.
- **`Propagar excepción manualmente`** Utilizando la palabra throw podemos propagar la excepción a un nivel más arriba,y dejar que la maneje otro catch, cabe aclarar que si se propago la excepción pero no se declaró ningún manejador se comunicará el error en ejecución.En caso de encontrarla se la maneja y se finaliza la ejecución del bloque en el que se encontraba el manejador especificado.
- **`Ejemplo de propagación manual`**

```
function funcion1() {
  throw new Error("¡Error en la función 1!");
}

function funcion2() {
  try {
    funcion1(); // Lanzar excepción
  } catch (error) {
    console.log("Excepción capturada en la función 2:", error.message);
    throw error; // Propagar la excepción hacia arriba
  }
}

function funcion3() {
  try {
    funcion2(); // Lanzar excepción
  } catch (error) {
    console.log("Excepción capturada en la función 3:", error.message);
    // No se propaga la excepción, se maneja aquí
  }
}

try {
  funcion3(); // Lanzar excepción
} catch (error) {
  console.log("Excepción capturada en el bloque principal:", error.message);
  // Aquí puedes realizar un manejo adicional o dejar que la excepción se propague hacia arriba
}
```

PYTHON

Al igual que javascript también posee manejo de excepciones por terminación y también Posee 2 circuitos en los que clasificar su manejo

Circuitos básicos de manejo de excepciones

try-except:	try-finally

Bloque
try:
    # Código susceptible a lanzar una excepción
    resultado = 10 / 0 # División por cero
except ZeroDivisionError:
    print("¡Error: División por cero!")
 	
Manejo de la excepción específica (división por cero)
	try:
	    # Código susceptible a lanzar una excepción
	    archivo = open("archivo.txt", "r")
	    # Operaciones con el archivo
	finally:
	    archivo.close()
	
	Bloque de código que siempre se ejecuta, independientemente de si se lanza una excepción o no

Circuitos avanzados de manejo de excepciones

try-except-finally	try-except con múltiples except

```python
try:
    # Código susceptible a lanzar una excepción
    resultado = int("ABC")
    # Conversión de una cadena no numérica a entero
except ValueError:
    # Manejo de la excepción específica (valor no numérico)
    print("¡Error: Valor no numérico!")
finally:
    print("Este bloque siempre se ejecuta.")
```

Bloque de código que siempre se ejecuta, independientemente de sise lanza una excepción o no

```python
try:
    # Código susceptible a lanzar una excepción
    lista = [1, 2, 3]
    print(lista[5])
    # Acceso a un índice inexistente
except IndexError:
    # Manejo de la excepción específica (índice fuera de rango)
    print("¡Error: Índice fuera de rango!")
except Exception:
    print("¡Error: Excepción general!")
```

Manejo de cualquier otra excepción no capturada por los bloques anteriores

¿Qué sucede si no se encuentra un manejador para la excepción ocurrida?

- **Plantear caso por defecto y manejar genéricamente** Deberíamos plantear el caso en que no se haya encontrado el manejador específico y poner uno como default como el caso de arriba que utiliza except Exception para capturar la excepción genérica de python .
- **Propagar excepción** Cuando no se encuentra un manejador para la excepción ocurrida en el bloque try , y no está especificada el caso por defecto de Exception, la misma se propaga un nivel más arriba, buscando un manejador para esa excepción, en caso de no encontrar ninguna excepción se comunica el error y finaliza la ejecución del programa.En caso de encontrarla se la maneja y se finaliza la ejecución del bloque en el que se encontraba el manejador especificado.

```python
def funcion1():
    # Código susceptible a lanzar una excepción
    raise ValueError("¡Error en la función 1!")

def funcion2():
    try:
        funcion1()  # Lanzar excepción
    except ValueError as e:
        print("Excepción capturada en la función 2:", str(e))
        raise  # Propagar la excepción hacia arriba

def funcion3():
    try:
        funcion2()  # Lanzar excepción
    except ValueError as e:
        print("Excepción capturada en la función 3:", str(e))
        # No se propaga la excepción, se maneja aquí

try:
    funcion3()  # Lanzar excepción
except ValueError as e:
    print("Excepción capturada en el bloque principal:", str(e))
    # Aquí puedes realizar un manejo adicional o dejar que la excepción se propaguehacia arriba
```

`Ejercicio D)`

Conclusión sobre los lenguajes: Realice una entrevista a un usuario/s experimentado de los lenguajes asignados con el fin de obtener una opinión acerca del lenguaje respecto de los temas tratados en el trabajo. Esta conclusión no debe superar una carilla. En caso de no encontrar un experto con facilidad pueden hacer una referencia a lo que ustedes consideran como aspectos fundamentales y relevantes del lenguaje (intentar privilegiar siempre la entrevista, incluso con docentes de la facultad.)

Conclusión de entrevista por lenguaje

Javascript

La característica principal del lenguaje es su versatilidad y flexibilidad en el desarrollo de aplicaciones. Una de sus fortalezas radica en su capacidad para manejar eventos y realizar programación asíncrona, lo que lo hizo altamente popular en aplicaciones interactivas y receptivas. Además, JavaScript cuenta con un amplio conjunto de funciones integradas y una sintaxis clara que facilita la escritura de código conciso y legible.
En términos de tipado, JavaScript se considera un lenguaje débilmente tipado debido a su capacidad para realizar conversiones automáticas de tipos. Esto puede proporcionar flexibilidad al permitir que una variable cambie su tipo durante la ejecución del programa. Sin embargo, esta flexibilidad puede requerir una mayor atención en el manejo de excepciones, ya que los errores de tipo pueden ocurrir en tiempo de ejecución si no se tiene cuidado.
El manejo de excepciones en JavaScript se realiza mediante el uso de bloques try-catch y bloques try-finally, permite capturar y manejar excepciones de manera efectiva. Sin embargo, debido a la naturaleza dinámica del lenguaje, es importante tener precaución al manejar las excepciones y garantizar la integridad del código.

Python

Su principal fortaleza es su simplicidad y legibilidad, su sintaxis clara y concisa permite escribir código de manera eficiente y fácilmente comprensible. En cuanto a la tipificación, Python se considera un lenguaje fuertemente tipado. Esto significa que las operaciones entre tipos incompatibles generan errores en tiempo de ejecución, lo que brinda seguridad y confiabilidad al desarrollar aplicaciones. Además, Python proporciona una amplia gama de estructuras de datos integradas, como listas y diccionarios, que facilitan el manejo y la manipulación de datos de manera eficiente.
El manejo de excepciones en Python es una característica fundamental del lenguaje. Python ofrece bloques try-except que permiten capturar y manejar excepciones de manera explícita. Esto brinda la oportunidad de detectar y corregir errores de manera anticipada, lo que es especialmente valioso en el desarrollo de aplicaciones robustas y confiables.
Además, Python cuenta con una amplia biblioteca estándar que incluye numerosas funciones y módulos incorporados. Esto proporciona una gran cantidad de herramientas para abordar una variedad de tareas, desde análisis de datos hasta desarrollo web y automatización de tareas.

`Ejercicio E)`

Conclusión sobre el trabajo: Realice una conclusión de una carilla como máximo, mencionando los aportes que le generó la realización del trabajo en comparación con sus conocimientos previos de los lenguajes asignados.

- La realización de este trabajo nos ha permitido adquirir nuevos conocimientos y ampliar nuestra comprensión de los lenguajes asignados, JavaScript y Python. A lo largo del proceso, conseguimos mejorar nuestro dominio en ambos lenguajes.
- Una de las áreas en las que hemos profundizado inicialmente es en las estructuras de control. Ahora sabemos cómo se implementan los bucles y las condiciones en cada lenguaje específicamente. Por ejemplo, en JavaScript su sintaxis flexible nos ofrece diversas opciones para controlar el flujo de ejecución, mientras que Python nos brinda claridad de su sintaxis, lo que facilita la escritura de código legible y estructurado.
- Comprendimos cómo utilizar los bloques try-catch en JavaScript y los bloques try-except en Python para capturar y gestionar las excepciones de manera efectiva, fue particularmente interesante el chequeo de instancias en javascript.
- Durante el desarrollo de este trabajo, también he adquirido un mejor entendimiento de la sintaxis específica de cada lenguaje. A través de los ejemplos y la búsqueda de información, hemos mejorado nuestra capacidad para escribir código en ambos lenguajes de manera más efectiva y eficiente.
- Exploramos el concepto de tiempo de ligadura en cada lenguaje, es decir, el momento en que se resuelven las referencias a variables y funciones durante la ejecución del programa. Esto me ha dado una comprensión más clara de cómo funcionan internamente JavaScript y Python y cómo interactúan las diferentes partes del código.
- Otro aspecto relevante que hemos investigado fue la detección de errores sintácticos y semánticos. Nos ha ayudado a identificar y solucionar problemas en el código de manera más eficiente.
- En cuanto a los tipos de datos y parámetros, hemos notado las diferencias entre ambos lenguajes. JavaScript, con su tipado débil, permite conversiones automáticas de tipos, mientras que Python es un lenguaje fuertemente tipado que requiere una correspondencia precisa entre los tipos de datos utilizados. Estas características me han dado una comprensión más profunda de cómo trabajar con los datos en cada lenguaje y cómo evitar errores relacionados con el tipo de datos.
- **En conclusión**, este trabajo nos ha proporcionado una base sólida en la sintaxis y semántica de ambos lenguajes, nos ha generado una mayor confianza en el uso de ambos para aprovechar al máximo sus capacidades y características.