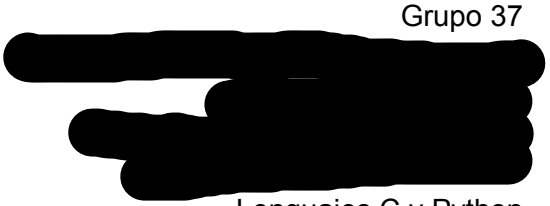
Trabajo práctico de Conceptos y Paradigmas de Lenguajes de Programación

Grupo 37



Lenguajes C y Python

Bibliografía usada:

https://docs.python.org/3/tutorial/errors.htm
https://learn.microsoft.com/en-us/cpp/c-language/?view=msvc-170

A. Escriba 2 fragmentos de código de mínimo 5 y máximo 10 líneas de extensión que permitan ejemplificar al menos dos criterios de evaluación en los lenguajes asignados. Puede citar un ejemplo para cada criterio elegido y compararlo entre los lenguajes, colocando una breve descripción textual para cada uno.

Python	C

try:
print(1/0)
except ZeroDivisionError as e:
print("no se puede dividir entre 0!")
list = [1,2,3,4,5]
	#include <stdio.h>
	struct Nodo{ //declaración de la estructura de lista enlazada
	  int dato;
	  struct Nodo *siguiente;
	};
	int main(){
	  int divisor = 0;
	  if(divisor == 0){
	  printf("No se puede dividir por cero\n");
	  }
	}

- Abstracción: En los fragmentos de código se puede observar la declaración de una lista tanto en C como en Python. En Python con simplemente ingresar los elementos que se van a agregar en la lista entre corchetes([]) y separarlos con comas(,), el lenguaje interpreta que estamos queriendo hacer una lista y se encarga de implementar la estructura.

  En cambio, para hacer una lista en C, tenemos que ocuparnos nosotros como programadores de expresar la estructura de la lista y la manejamos mediante punteros. A su vez, el manejo de la lista queda también del lado del programador. Por ejemplo, para agregar un elemento al final de la lista debemos desplazarnos mediante los punteros hasta el último elemento y agregarlo en esa posición, mientras que en Python podemos hacerlo simplemente con lista.append().

- Simplicidad y legibilidad: La legibilidad del código en Python contrasta con la complejidad de C. En Python, la declaración de listas es simple y legible, con una sintaxis clara y concisa. En cambio, en C, la construcción de listas requiere el uso de palabras reservadas y una estructura más robusta, lo que puede resultar menos legible y más propenso a errores. En resumen, Python ofrece una sintaxis más amigable y comprensible, mientras que en C, la complejidad del lenguaje puede dificultar la legibilidad del código.

- Confiabilidad: Python tiene un manejo de excepciones que viene incorporado con el lenguaje, mientras que en C debemos ocuparnos nosotros de implementar algo que haga algo similar. Por otro lado, otra cosa que los hace diferentes en términos de

confiabilidad es que C hace un chequeo de tipos en compilación, mientras que en Python se hace en ejecución.

B. Cite 2 ejemplos de código (mínimo 5, máximo 15 líneas) para cada lenguaje asignado que permitan analizar A NIVEL SINTÁCTICO 2 estructuras de control de manera comparada entre los lenguajes. Coloque una breve descripción a cada ejemplo.

Python	C

`input= "s"`
`if usuario == "S" or usuario == "s":`
`  print("El usuario eligió S")`
`  print("Verdadero")`
`elif usuario == "N" or usuario == "n":`
`  print("Falso")`
`else:`
`  print("Valor incorrecto")`
	`#include <stdio.h>`
	`int main() {`
	`  char input = 's'`
	`  if (input == 's' || input == 'S'){`
	`  printf("El usuario eligió S\n");`
	`  printf("Verdadero\n");`
	`  }`
	`  else if (input == 'n' || input == 'N')`
	`  printf("Falso\n");`
	`  else`
	`  printf("Valor Incorrecto\n");`
	`}`

- La sintaxis abstracta en ambos lenguajes es similar, se introduce la palabra reservada “if”, la condición y el símbolo que da inicio al bloque a ejecutar en caso de que la condición sea verdadera. En términos de sintaxis concreta, sí que son bastante diferentes. En primer lugar, el bloque a ejecutar en C se encierra entre llaves (cuando tiene más de una línea) mientras que en Python se indenta. Por otro lado, en Python, la estructura “elif” se utiliza para lo que en C se conoce como “else if”, lo cual ofrece una sintaxis más clara y concisa.. En tanto a las diferencias de sintaxis pragmática, es más legible y claro el uso del “or” de Python que el “||” de C para definir la suma lógica.

Python	C

`x = 10`
`while x > 0:`
`  print(x)`
`  x-=1`
	`#include <stdio.h>`
	`int main(){`
	`  int x = 10;`
	`  while (x > 0) {`
	`  printf("%d\n", x);`
	`  x--;`
	`  }`
	`}`

- Se puede apreciar una sintaxis abstracta similar en ambos lenguajes. Se inicializa el bloque con la palabra reservada “while”, seguida por la condición de corte y luego el bloque. No obstante, tienen diferencias a nivel concreto. C encierra los bloques entre corchetes mientras que en Python simplemente se le da comienzo al bloque con “:” y se usa la indentación para determinar qué líneas abarca. A su vez, las sentencias de C se finalizan con “;” mientras que Python no usa ningún carácter con ese fin. En términos de sintaxis pragmática, C no incluye por defecto una forma de mostrar por pantalla ni leer del teclado, por lo que se debe incluir la línea “#include <stdio.h>” con ese fin, mientras que en Python dicha funcionalidad viene por defecto.

C. Cite un ejemplo de código (5 líneas mínimo, 10 líneas máximo) para cada lenguaje asignado donde pueda distinguirse con claridad chequeos de semántica estática y dinámica a la hora de ejecutar/interpretar código. Debe poder entenderse con claridad, más allá de la naturaleza del lenguaje, distintos tipos de errores que se controlan y el momento en que dichos chequeos se realizan hasta poder ejecutar propiamente el programa.

C	Python

int main(){
  int a = 1;
  a = "hola";// error semántica estática
  int arr[2];
  arr[0] = 1;
  arr[1] = 2;
  print(arr[2]);// error semántica dinámica
}
	texto = "Hello World"
	numero = int(texto)
	print(numero)
	#Intento de llamar a una función que no existe
	print(funcion_inexistente())

- En el ejemplo en C, se está intentando asignar una string a una variable previamente declarada como int. Este error es de semántica estática y será detectado por el compilador. Por otro lado, en el fragmento de código se puede observar la creación de un array de 2 elementos y el posterior intento de acceder al elemento en la posición 2 (que está fuera de su rango). Este es un error que será detectado al hacer el chequeo de semántica dinámica.

- En Python, los errores de semántica estática no ocurren porque Python es un lenguaje de tipado dinámico. Esto significa que las comprobaciones de tipo y otros análisis que suelen ser estáticos no se realizan hasta que se ejecuta el código. Por otro lado, en el ejemplo brindado se pueden observar dos errores detectados en el chequeo de semántica dinámica. Por un lado, en la línea 2 se está intentando hacer un casteo que no es válido (de string a int). Por el otro, en la 5 se llama a una función que no está declarada.

D. Defina un ejemplo de código (5 líneas mínimo,15 líneas máximo) en cada lenguaje seleccionado donde se visualicen los distintos tipos de variables que soporta el lenguaje en cuanto al tiempo de vida, y realice una tabla donde se sitúen todos los identificadores junto con los valores de sus atributos tal como se vió en la práctica correspondiente.

Ejemplo de C:

```c
#include <stdio.h>
int variable_global = 10;  // Variable global
int *puntero;
void mi_funcion() {
    static int variable_estatica;  // Variable estática local
    int variable_automatica = 20;  // Variable automática local
    int variable_global = 5;
    puntero = &variable_global;  // Puntero que apunta a la variable global.
}
int main() {
    mi_funcion();
    printf("Variable global fuera de la función: %d\n", variable_global);
}
```

Identificador	L-Valor	R-Valor	Alcance	Tiempo de vida
variable_global(2)	automática	10	3-7 10-13	2-13
mi_funcion			5-13	4-9
Variable_estática	estática	0	6-9 13>	<1-13>
variable_automatica	automática	20	7-9	6-9
puntero	automática	null	4-13	8-9
*puntero	dinámica	&variable_global(2)	8-9	8-9
variable_global(7)	automática	5	8-9	7-9
main			11-13	10-13

Ejemplo en Python:

```
1. x = 11 // Variable global
2. def main():
3.   x = 1 // Variable local
4.   y = 5
5.   print(x)
6. def mi_function():
7.   global x
8.   x = 7
```

Identificador	L-Valor	R-Valor	Alcance	Tiempo de vida
x(1)	dinámica	11,7	2-3 6-8	1-8
main			3-8	2-6
x(3)	dinámica	1	4-6	3-6
y	dinámica	5	5-6	4-6
mi_function			7-8	6-8