Trabajo Práctico N° 1
Paradigmas de Programación

Dr. Pablo Javier Vidal Unidad 1

Ejercicio 1.

Responder las siguientes preguntas

1. **El breve estudio histórico presentado en clase mediante una guía de evolución de lenguajes, no menciona todos los principales lenguajes de programación existentes hoy en día (solo los que han sido particularmente influyentes, en opinión de los autores seleccionados). Si se ha omitido uno de sus lenguajes favoritos o de uso cotidiano, explique por qué cree que es lo suficientemente importante como para incluirlo y muestre dónde encaja su lenguaje seleccionado en dicha figura.**

JavaScript: Esencial para el desarrollo web moderno, permite interacción dinámica en sitios y aplicaciones web, así como en aplicaciones móviles y servidores.

C#: Vital en el ecosistema Microsoft, utilizado para crear aplicaciones de escritorio, web y juegos, especialmente con Unity.

PHP: Fundamental en el desarrollo web en el lado del servidor, genera contenido dinámico y trabaja con bases de datos.

R: Crucial en la ciencia de datos y análisis estadístico, ofrece una rica biblioteca para el procesamiento y visualización de datos.

Swift: Desarrollado por Apple, es esencial para crear aplicaciones en iOS, macOS, watchOS y tvOS, reemplazando a menudo a Objective-C.

2. **Fortran y Cobol son dos lenguajes de hace muchas décadas, muy viejos en cuanto a transparencia al cliente y al programador. ¿Puede indicar porque aún siguen siendo utilizados? ¿Qué características presentan los dos lenguajes teniendo en cuenta lo visto en clase?.**

Fortran: Se sigue utilizando en la actualidad para comparar programas y clasificar a los superordenadores más potentes del mundo.

Cobol: sigue siendo relevante en la actualidad debido a su uso extensivo en sistemas críticos y su estabilidad, eficiencia y seguridad comprobadas.

3. **¿Qué estructura presenta Assembler con respecto a las características que definen a un lenguaje (nivel del lenguaje, sistema de tipos, ligaduras, etc).**

Assembler			
nivel del lenguaje	*máquina*	*bajo nivel*	*alto nivel*
sistema de tipos	*estático* | *dinámico*	*fuerte* | *débil*	*nominal* | *estructural*
ligaduras	*estática* | *dinámica*		

4. ¿Qué características presenta el lenguaje Python con respecto a su nivel del lenguaje, sistema de tipos, ligaduras, entre otras.

Python			
nivel del lenguaje	*máquina*	*bajo nivel*	*alto nivel*
sistema de tipos	*estático* | *dinámico*	*fuerte* | *débil*	*nominal* | *estructural*
ligaduras	*estática* | *dinámica*		

5. ¿Python es compilado o interpretado? Describir el proceso que realiza Python desde el momento que se toma el código fuente hasta lograr ser ejecutado en la máquina.

Python es un lenguaje interpretado. Cuando ejecutas un script de Python:

- *El intérprete analiza el código fuente en busca de errores.*
- *El código se compila a un formato intermedio llamado bytecode.*
- *El intérprete ejecuta el bytecode línea por línea.*
- *Durante la ejecución, se crean objetos y se administra la memoria.*
- *El intérprete produce resultados como salida en la consola o archivos.*

Ejercicio 2.

Indicar cuál de las siguientes afirmaciones referentes a la Programación Declarativa (PD) e Imperativa (PI) es falsa o verdadera. Justifique la respuesta.

1. Las instrucciones de un PD son un conjunto de inferencias lógicas. *falsa*
2. Un programa en PD se corresponde con las instrucciones para resolver un problema. *verdadera*

En Programación Declarativa, un programa describe "qué" se quiere lograr en lugar de "cómo" lograrlo.

Ejercicio 3.

Distinga entre:

1. Tipos, Ligadura estática de tipos y ligadura dinámica de tipos.

- *Tipos: Representan categorías de datos con propiedades y operaciones asociadas.*
- *Ligadura Estática de Tipos: La verificación de tipos se realiza en tiempo de compilación. Los tipos se asignan a las variables y se verifican antes de la ejecución.*
- *Ligadura Dinámica de Tipos: La verificación de tipos se realiza en tiempo de ejecución. Los tipos se determinan durante la ejecución de acuerdo con los valores asignados a las variables.*

2. Coacción, error de tipos y comprobación de tipos.

- *Coacción (Type Casting): Conversión explícita entre tipos de datos compatibles.*
- *Error de Tipos (Type Error): Ocurre cuando se operan valores de tipos incompatibles.*
- *Comprobación de Tipos (Type Checking): Proceso para garantizar operaciones coherentes entre tipos en el programa. Puede ser estática (compilación) o dinámica (ejecución).*

3. Compatibilidad de tipos nominal y compatibilidad de tipos estructural.

- *Compatibilidad de Tipos Nominal: Se basa en nombres o subtipos explícitos.*
- *Compatibilidad de Tipos Estructural: Se basa en la estructura interna sin importar los nombres.*

Ejercicio 4.

En tiempo de ejecución, cuando entramos en un bloque las variables locales ya no contienen los valores que almacenaban la última vez que fue ejecutado dicho bloque. ¿Por qué ocurre esto?

Las variables locales en un bloque se reinician cuando entramos en él debido a cómo se maneja la memoria y el alcance en los lenguajes de programación. Esto ayuda a administrar la memoria de manera eficiente, garantizar un estado predecible y mantener la limitación de alcance de las variables. Al entrar en un nuevo bloque, se asigna espacio en la memoria para las variables locales, y al salir del bloque, ese espacio se libera y las variables ya no son accesibles ni conservan sus valores anteriores.

Ejercicio 5.

Dado el siguiente esqueleto de programa escrito en C, dibuja un rectángulo alrededor de cada uno de sus bloques, etiquetalos con las letras A, B, C, etc. y responde a las siguientes preguntas.

```
int x, y, z;

fun1()
{
  int j, k, l;
  {
    int m, n, x;
     A
  }
}

fun3()
{
  int y, z, k;
    B
}

void main()
{  C  }
```

1.Mencione un bloque que esté anidado dentro de otro bloque.
A

2.En qué ámbitos son accesibles a la vez las variables globales x, y, z?
en todos los ámbitos

3.Qué variables son accesibles desde la función main?
(x, y, z)

4.Son k de fun1 y k de fun3 la misma variable? Razona la respuesta.
No, k en fun1() y k en fun3() no son la misma variable, ya que son variables locales con el mismo nombre

5.Son las variables globales y y z accesibles desde fun3?. Justi que la respuesta.
Sí, las variables globales y y z son accesibles desde fun3(), ya que están definidas en el ámbito global y, por lo tanto, son visibles desde cualquier función en el programa.

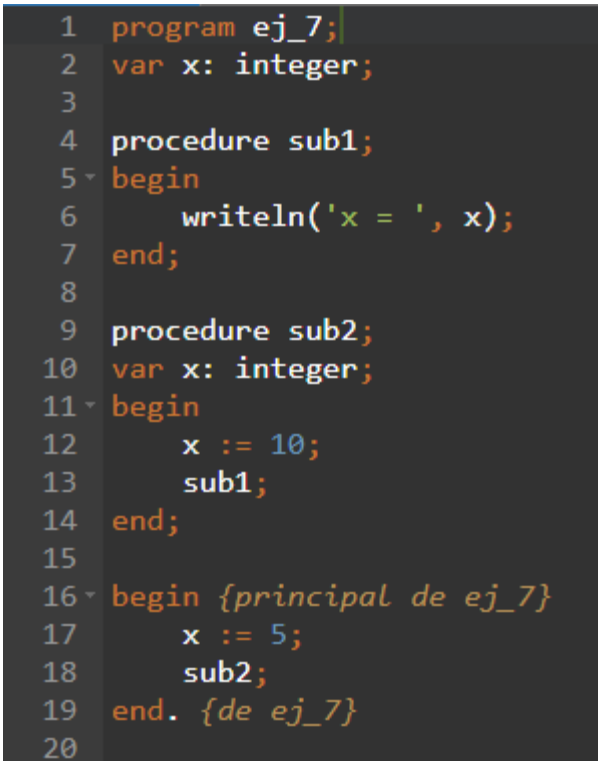
6.En qué bloques j hace referencia a una variable local? ¿En qué bloques j es una variable no local?
En fun1(), j hace referencia a una variable local dentro de ese bloque.
En fun3(), no hay una variable local llamada j. En este bloque, no se hace referencia a j.

7.En qué bloque o bloques pueden usarse a la vez m y k?
A

Ejercicio 6.

Indica qué valor de x se imprime en el procedimiento sub1 suponiendo tanto

ámbito estático como ámbito dinámico.



```
1  program ej_7;
2  var x: integer;
3
4  procedure sub1;
5  begin
6      writeln('x = ', x);
7  end;
8
9  procedure sub2;
10 var x: integer;
11 begin
12     x := 10;
13     sub1;
14 end;
15
16 begin {principal de ej_7}
17     x := 5;
18     sub2;
19 end. {de ej_7}
20
```

Estático: x == 5
Dinámico: x == 10

Ejercicio 7.

Indicar cuál de las siguientes afirmaciones es cierta. Justificar su respuesta.

1. **Un analizador léxico (scanner) es un programa que divide una secuencia de caracteres (el programa) en una secuencia de componentes sintácticos primitivos: las instrucciones.**

2. **La compatibilidad de tipos, el alcance de las variables y la signatura de las funciones (coincidencia del número de parámetros en una llamada con los parámetros formales) forman parte de la semántica estática del lenguaje.**

La afirmación cierta es: 2

Analizador Léxico: Un analizador léxico divide un programa en componentes sintácticos primitivos llamados tokens, que son las unidades básicas del lenguaje.
Semántica Estática: La semántica estática se refiere a propiedades del programa analizadas en tiempo de compilación, incluyendo la compatibilidad de tipos, el alcance de las variables y la signatura de las funciones. Estos aspectos garantizan la corrección antes de la ejecución del programa.

Ejercicio 8.

Indicar cuál de las siguientes afirmaciones es cierta. Justificar su respuesta.

1. **Una desventaja de los lenguajes de programación declarativos es que, para resolver un mismo problema, se requieren en general más líneas de código que usando otros lenguajes de programación más convencionales, como Python.**

2. **Al ser de mayor nivel, los programas declarativos resultan más fáciles de mantener que los correspondientes programas imperativos.**
3. **Los lenguajes declarativos carecen de aplicaciones prácticas.**

Afirmación 1: Incorrecta. No es cierto que los lenguajes declarativos siempre requieren más líneas de código que los imperativos para resolver un problema. La brevedad del código depende del problema y la implementación.

Afirmación 2: Cierta. Los programas declarativos suelen ser más fáciles de mantener debido a que se centran en "qué" hacer en lugar de "cómo" hacerlo, lo que mejora la legibilidad y comprensión.

Afirmación 3: Incorrecta. Los lenguajes declarativos tienen muchas aplicaciones prácticas en áreas como bases de datos, inteligencia artificial y análisis de datos, como SQL y Prolog.

Ejercicio 9.

Indica cuál de las siguientes afirmaciones sobre la programación declarativa en comparación con la programación imperativa es VERDADERA o FALSA:

1. **Mayor potencia expresiva.** *verdadera*
2. **Mantenimiento más simple** *verdadera*
3. **Menor tamaño del código producido.** *falsa*
4. **Establecer el cómo proceder** *falsa*

Ejercicio 10.

Indica cuál de las siguientes afirmaciones es VERDADERA o FALSA.

El lenguaje ensamblador:

1. **Más cerca del lenguaje máquina que de los lenguajes de alto nivel** *verdadera*
2. **Más cerca de los lenguajes de alto nivel que del lenguaje máquina** *falsa*
3. **No es un lenguaje de programación** *falsa*

Ejercicio 11.

Clasifica los siguientes lenguajes para saber si son compilados/interpretados/híbridos. Lenguajes: Fortran, JavaScript, Mathematica, C,Java, Perl, Pascal, Haskell, Scala, PHP.

Compilados	*Fortran, C, Java, Pascal, Haskell, Scala*
Interpretados	*JavaScript, Mathematica, Perl, PHP*
Híbridos	

Ejercicio 12.

Indica cuál de las siguientes afirmaciones es VERDADERA o FALSA.

En un lenguaje débilmente tipado:

1. **Un valor de un tipo puede ser tratado como de otro tipo** *verdadera*
2. **Un valor de un tipo nunca puede ser tratado como de otro tipo** *falsa*
3. **Un valor de un tipo puede ser tratado como de otro tipo siempre que se realice una conversión de forma explícita** *falsa*
4. **Las anteriores respuestas no son correctas** *falsa*

Ejercicio 13.

Imperativo, declarativo y orientado a objetos son:

1. **Modos de compilar el código fuente de un programa de ordenador** *falsa*
2. **Modos de definir el pseudocódigo de un programa de ordenador** *falsa*
3. **Paradigmas de programación** *verdadera*
4. **Las anteriores respuestas no son correctas** *falsa*

Ejercicio 14.

En base a lo aprendido sobre ligaduras, respondas las siguientes preguntas:

1. **¿Qué es una ligadura? ¿Cómo se pueden clasificar los tiempos de ligadura? ¿A qué características o criterios de evaluación, en general, afecta el tiempo de una ligadura?**

- *Una ligadura es la asociación entre dos cosas*
- *Dependiendo del momento de asociación la ligadura puede ser estática(tiempo de compilación) o dinámica (tiempo de ejecución)*
- *afecta al tiempo de compilación y al tiempo de ejecución*

2. **El concepto de ligadura, está vinculado a la sintaxis o la semántica de un lenguaje de programación? Justifiqué y ejemplifiqué.**

La ligadura es un concepto semántico porque determina el significado y la referencia de los identificadores en un programa

3. **¿Cuál es la diferencia entre una ligadura implícita y una explícita? Brindar ejemplos de cada una de ellas.**

Ligadura Implícita: El lenguaje de programación asocia automáticamente un nombre con una entidad sin intervención directa del programador. Ejemplo: Asignación de valores a variables en muchos lenguajes.
Ligadura explícita: El programador establece intencionalmente la relación entre un nombre y una entidad mediante acciones específicas en el código. Ejemplo: Asignación manual de una dirección de memoria a un puntero en C.

Ejercicios Prácticos

Ejercicio 15.

Implementar los siguientes enunciados en Python utilizando los conceptos básicos de programación de recursividad, sin el uso de módulos extras.

1. Función que devuelva el n-ésimo número de la sucesión de Fibonacci. ¿Qué valor devuelve para n=10? y para n=10000?

python main.py -fibonacci <n>
cuando n == 10, resultado == 55
cuando n == 10000, resultado == error

2. La función potencia(b,n), devuelve el valor de elevar b a la potencia n.

python main.py -potencia <b> <n>

3. Escribir una función recursiva que reciba como parámetros dos cadenas de caracteres *a* y *b*, una lista *X* vacía y el índice de inicio. La función devuelve en *X* una lista con las posiciones en donde se encuentra *b* dentro de *a*. Ejemplo:

```
>>> posiciones_de("Un tete a tete con Tete", "te",[],0)
[3, 5, 10, 12, 21]
```

python main.py -posiciones_de <"a"> <"b"> <"[]"> <i>

4. Escribir una función booleana recursiva llamada Vectores Iguales que reciba dos listas como parámetros y devuelva TRUE si son iguales (mismos elementos en el mismo orden), o FALSE en caso contrario.

python main.py -vectores_iguales "[1, 2, 3, 4, 5]" "[1, 2, 3, 4, 5]"

5. Desarrollar una función que permita recursivamente armar el la n-esima linea del triangulo de pascal

Ejemplo:

```
>>> pascal(5)
[1, 4, 6, 4, 1]
```

python main.py -pascal_line <n>

6. Escriba una función recursiva llamada Invertir que, dada una lista L, la invierta.

python main.py -invertir <"list">

7. En una empresa de entrega de correo los trabajos no se hacen como en un bucle iterativo donde todo se va entregando secuencialmente. Para ello existen dos reglas básicas.

Si el número de paquetes es mayor a 1 estamos ante la presencia de un administrador el cual puede nombrar dos trabajadores y dividir su trabajo entre ellos

Si el número de paquetes es mayor a 1 él es un trabajador y tiene que entregar el paquete a la casa que le ha sido asignada.

Se pide al alumno desarrollar una función recursiva que ante una lista de posibles casas donde entregar, permita informar la entrega del paquete cuando lo tenga el trabajador. Ejemplo

```
>>> las_casas = ["Casa de Pablo", "Casa de Caro",  "Casa de Caty", "Casa de Luna"]
>>> entrega_de_paquetes(las_casas)
        Entrego el paquete en Casa de Pablo
        Entrego el paquete en Casa de Caro
        Entrego el paquete en Casa de Caty
        Entrego el paquete en Casa de Luna
```

python main.py -entrega_de_paquetes "['Casa de Pablo','Casa de Caro','Casa de Caty','Casa de Luna']"

8. Escribir una función que simule el siguiente experimento: Se tiene una rata en una jaula con 3 caminos, entre los cuales elige al azar (cada uno tiene la misma probabilidad), si elige el 1 luego de 3 minutos vuelve a la jaula, si elige el 2 luego de 5 minutos vuelve a la jaula, en el caso de elegir el 3 luego de 7 minutos sale de la jaula. La rata no aprende, siempre elige entre los 3 caminos con la misma probabilidad, pero quiere su libertad, por lo que recorrerá los caminos hasta salir de la jaula.

La función debe devolver el tiempo que tarda la rata en salir de la jaula. Ejemplo:

```
>>> ratatuille(randint(1,3))
        Va por camino 1
        Va por camino 1
        Va por camino 2
        Va por camino 2
        Va por camino 3
        Tiempo total: 23
```

python main.py -ratatuille

9. En la vereda de una calle con adoquines unos niños juegan al tejo. Para esto enumeran los adoquines de la siguiente forma:

Los movimientos permitidos del juego son:

Al principio del juego los niños se ubican en el adoquín 1.

De un adoquín numerado *i* se puede saltar al adoquín numerado *i* + 1.

De un adoquín numerado *i* se puede saltar al adoquín numerado *i* + 2 (sin

pasar por el adoquín *i* + 1). Por ejemplo, el número de caminos posibles para n=3 es 3 y son los siguientes: (0,1,2,3), (0,2,3) y (0,1,3).

Escriban una función recursiva llamada *Allways*(*n*) que calcule el número de caminos posibles para alcanzar un adoquín objetivo numerado con n (mayor que cero).

python main.py -Allways <n>