Aspectos conceptuales

1. ¿Qué es un programa?

Un programa de computadora es un conjunto de instrucciones u órdenes dadas a la máquina que producirán la ejecución de una determinada tarea. En esencia, un programa es un medio para conseguir un fin. Tras la decisión de desarrollar un programa, el programador debe establecer el conjunto de especificaciones que aquél debe contener: entrada, salida y algoritmos de resolución; éstos incluirán las técnicas para obtener las salidas a partir de las entradas.

2. ¿Por qué cree conveniente separar en fases el proceso de desarrollo de software?

El estilo de vida intensivo en tecnología induce al software a ser una parte integral de la rutina diaria en el siglo XXI. Hoy en día, es casi imposible imaginar una actividad que no esté impulsada por algún tipo de proceso relacionado con la computadora. Experimentar una compilación de software personalizada puede ser abrumador para casi cualquier organización, y más aún si no tiene experiencia técnica.

Al profundizar, el desarrollo de productos de software es un proceso altamente organizado con procedimientos precisos y pasos estrictamente definidos conocidos como ciclo de vida de desarrollo de software (SDLC). Siempre que necesite un sistema sofisticado, una suite de software o una aplicación web o móvil para el usuario final, la entrega de un proyecto, además de todos los otros factores importantes, depende en gran medida de un conjunto de procesos practicados por el equipo de desarrollo: El ciclo de vida del desarrollo de software como una colección de reglas y prácticas ayuda a conectar a los miembros del equipo de tecnología, a los no tecnológicos y a las partes interesadas del proyecto para transformar su idea en un producto o solución de software.

En la práctica, estructura el trabajo de los equipos de desarrollo permitiéndoles cumplir con los requisitos del proyecto, cumplir con los plazos y mantenerse dentro del presupuesto.

3. ¿Qué dice el teorema de la programación?

Primero, que toda estructura posee un solo punto de entrada y uno solo de salida de la misma.

Segundo, que existen varios caminos desde la entrada hasta la salida.

Tercero, que todas las instrucciones pueden ser ejecutables en algún momento y no existen bucles infinitos o sin fin.

4. ¿Para qué sirven las bibliotecas?

Dan soporte a las operaciones mas frecuentes y con solo llamarlas permiten realizar operaciones sin necesidad de re-escribir su código.

5. Compare y relacione el concepto de compilar con el de interpretar.

Los lenguajes interpretados son multiplataforma, recordemos que una plataforma es un sistema operativo que ejecuta programas, por lo que son más flexibles pero se requiere de un intérprete (programa informático que analiza y ejecuta otros programas) para traducirlo y que la máquina lo reconozca. Todo esto sucede al instante ya que no se guarda la traducción del intérprete, es decir, se traduce al momento cada línea de código de forma ordenada cada vez

que se ejecuta el programa (por lo que lo hace un poco más lento en comparación a los compilados debido a la sobrecarga al momento de la ejecución). A pesar de esta diferencia de tiempo, los lenguajes interpretados son más populares en el desarrollo de programas que se modifican constantemente, ya sea para agregar o corregir una parte del código. Si bien el aspecto de parecer más lentos puede ser algo negativo, también tiene su punto a favor ya que, a diferencia de los lenguajes compilados que requieren de compilar y esperar a que el programa se haya terminado para saber si existen errores, en los lenguajes interpretados se sabe de inmediato si existe un error y su localización ya que se ejecuta linea por linea, así se puede corregir y modificar el error.

Por otro lado los lenguajes compilados están preparados para ejecutarse inmediatamente ya que durante la compilación se tradujo todo a un lenguaje que la máquina entiende (lenguaje máquina) y por ello suelen ser más rápidos. De modo que, gracias al compilador se traduce el código fuente, se crea un archivo ejecutable y desde este no se puede acceder a su código fuente. Dependiendo del caso, esto podría ser o no algo positivo. Esto se debe a que una vez compilado el programa ya no es necesario el código fuente para poder ejecutarlo. Al obtener el archivo objeto puedes ejecutar el programa sin tener que compilar cada vez que lo ejecutes. Otra diferencia radica en que estos no son multiplataforma (solo sirven para una plataforma específica) y de querer ejecutarlo en otra plataforma, se debe compilar de nuevo, lo que le suma un paso extra.

6. Clasifique los lenguajes de programación según su nivel.

Lenguaje de máquina: Los lenguajes de máquina son aquellos que están escritos en lenguajes directamente inteligibles por la máquina (computadora), ya que sus instrucciones son cadenas binarias (cadenas o series de caracteres dígitos 0 y 1) que especifican una operación, y las posiciones (dirección) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina. El código máquina es el conocido código binario. Las instrucciones en lenguaje de máquina dependen del hardware de la computadora y, por tanto, diferirán de una computadora a otra.

Las **ventajas** de programar en lenguaje de máquina son las posibilidades de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior, lo que supone una velocidad de ejecución superior a cualquier otro lenguaje de programación.

Los **inconvenientes** -en la actualidad- superan a las ventajas, lo que los hace prácticamente no recomendables. Estos inconvenientes son: (1) Dificultad y lentitud en la codificación; (2) Poca fiabilidad; (3) Gran dificultad de verificar y poner a punto los programas; (4) Los programas sólo son ejecutables en el mismo procesador.

Lenguaje de bajo nivel (ensamblador): Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador. Las instrucciones en lenguaje ensamblador son instrucciones conocidas como nemotécnicos. Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: SUM (ADD), RES (SUB), DIV (DIV), etc. Una instrucción típica de suma sería: ADD P, T, A. Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la computadora -en esto se diferencia esencialmente del lenguaje máquina-, sino que requiere una fase de traducción al lenguaje de máquina. El programa original escrito en lenguaje ensamblador se denomina programa fuente y el programa traducido en lenguaje de máquina se conoce como programa objeto, directamente inteligible por la computadora.

Los lenguajes ensambladores presentan la **ventaja** frente a los lenguajes de máquina de su mayor facilidad de codificación y, en general, su velocidad de cálculo.

Los **inconvenientes** más notables de los lenguajes ensambladores son que hoy en día tienen aplicaciones muy reducidas en la programación y se centran en aplicaciones de tiempo real, control de procesos y de dispositivos electrónicos, etc.

Lenguajes de alto nivel: Los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en un lenguaje de alto nivel es independiente de la máquina; esto es, las instrucciones del programa de la computadora no dependen del diseño del hardware o de una computadora en particular. En consecuencia, los programas escritos en lenguajes en alto nivel son transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de plataformas (al contrario que los programas en lenguaje máquina o ensamblador, que sólo se pueden ejecutar en un determinado tipo de computadora).

Los lenguajes de alto nivel presentan las siguientes **ventajas**: (1) El tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes; (2) La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos; (3) Se utilizan nombres en las instrucciones, tales como read, write, print, open, etc.; (4) Las modificaciones y puestas a punto de los programas son más fáciles; (5) El costo de los programas se reduce; (6) Son transportables.

Los **inconvenientes** son: (1) Incremento del tiempo de puesta a punto, al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo; (2) No se aprovechan los recursos internos de la máquina, que se explotan mucho mejor en lenguajes de máquina y ensambladores; (3) Necesidad de una mayor capacidad de memoria; (4) El tiempo de ejecución de los programas es mucho mayor.

7. ¿Qué es un tipo de datos? Indique los que conozca y explique para que sirve.

El tipo de dato informático es un atributo de una parte de los datos que indica al ordenador (y/o al programador) algo sobre la clase de datos sobre los que se va a procesar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

Datos numéricos: El tipo numérico es el conjunto de los valores numéricos. Éstos pueden representarse en dos formas distintas:

- Tipo numérico entero (integer).
- Tipo numérico real (real).

Datos lógicos (booleanos): El tipo lógico -también denominado booleano- es aquel dato que sólo puede tomar uno de dos valores: verdadero (true) o falso (false). Este tipo de datos se utiliza para representar las alternativas (sí/no) a determinadas condiciones. Por ejemplo, cuando se pide si un valor entero es par, la respuesta será verdadera o falsa, según sea par o impar.

Datos tipo carácter y tipo cadena: El tipo carácter es el conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato tipo carácter contiene un solo carácter. Los caracteres que reconocen las diferentes computadoras no son estándar; sin embargo, la mayoría reconoce los siguientes caracteres alfabéticos y numéricos:

- caracteres alfabéticos (a, b, c, ...z),
- caracteres numéricos (0, 1, 2, ...9)
- caracteres especiales (+, -, .../\)

Algunos lenguajes tienen datos tipo cadena. Una cadena (string) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación.

8. ¿Qué tipos de estructuras conoce? Ejemplifique.

ESTRUCTURA SECUENCIAL

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el final del proceso.

Ejemplo: Dado el valor de la hora y la cantidad de horas trabajadas por un empleado, calcular su sueldo.

En seudocódigo:

```
Comienzo
      Ingresar, "Nro de empleado"
      Leer, nro_emp
      Ingresar, "Cantidad de horas trabajadas"
      Leer, cant_hor
      Ingresar, "Valor de la hora trabajada"
      Leer, val_hor
      sdo = cant_hor * val_hor
      Imprimir "Empleado", nro_emp, "cobra", sdo, "pesos"
Fin
```

ESTRUCTURA CONDICIONAL

Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí se suelen denominar estructuras de decisión o alternativas. En las estructuras selectivas se evalúa una condición y en función del resultado de la misma se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. Puede darse tres casos:

1) Condicionales con salida por el verdadero de la condición especificada.

Ejemplo: Ingrese dos lados de un triangulo, indique si son iguales y por lo tanto que el triangulo no puede ser escaleno.

En seudocódigo:

```
Comienzo
      Ingresar, "Valor del primer lado"
      Leer, L1
      Ingresar, "Valor del segundo lado"
      Leer, L2
      Si L1 = L2 Entonces
            Imprimir "Son iguales"
            Imprimir "Triángulo no escaleno"
      FinSi
Fin
```

2) Condicionales con salida por el verdadero y por el falso de la condición especificada.

Ejemplo: Ingresar dos valores, sumarlos si son iguales y multiplicarlos si son distintos.

En seudocódigo:

```
Comienzo
      Ingresar, "Primer valor"
      Leer V1
      Ingresar "Segundo valor"
      Leer V2
      Si V1=V2 Entonces
            C = V1 + V2
            Imprimir "Son iguales. La suma es", C
      De Lo Contrario
            C = V1 + V2
            Imprimir "Son distintos. Producto es", C
      FinSi
Fin
```

3) Condicional case o switch

Este tipo de condicional sólo se da cuando una variable puede tomar varios valores enteros en general y por cada una de esos valores tomar distintas alternativas de acción.

Ejemplo: Ingresar el número de empleado y la categoría a la que pertenece (son 4). Calcule cuantos empleados hay en cada una de ellas. Los datos finalizan con emp = 0.

En seudocódigo:

```
Comienzo
      Ingresar, "Ingrese empleado y categoría"
      Leer, emp, cat
      Hacer Hasta emp = 0
            Seleccionar Caso cat
                  Caso 1: C1=C1+1
                  Caso 2: C2=C2+1
                  Caso 3: C3=C3+1
                  Caso 4: C4=C4+1
            FinSelección
            Ingresar, "Ingrese empleado y categoría"
            Leer, emp, cat
      Repetir
      Imprimir "Empleados Categoría 1", C1
      Imprimir "Empleados Categoría 2", C2
      Imprimir "Empleados Categoría 3", C3
      Imprimir "Empleados Categoría 4", C4
Fin
```

ESTRUCTURAS ITERATIVAS

Cuando se utiliza un bucle para sumar una cantidad de números, se necesita saber cuántos números se han de sumar. Por eso necesitaremos conocer algún medio para detener el bucle. El bucle podrá terminar incorporando cualquiera de estas condiciones:

- Desde 1 Hasta N
- HastaQue (variable) sea (variable)

La condición del bucle normalmente se indica al principio o al final de este, de esa manera podemos considerar tres tipos de instrucciones o estructuras repetitivas:

- Mientras (while)
- Repetir (repeat)
- Desde/Para (for)

Desde 1 Hasta N (ciclos repetitivos exactos)

Ejemplo: Dados los sueldos de N empleados, determinar el total a pagar.

En seudocódigo:

```
Comienzo
      Ingresar, "Ingrese cantidad de empleados"
      Leer, N
      Para A <- 1 Hasta N Hacer
            Ingresar, "Ingrese sueldo"
            Leer, sdo
            tot = tot + sdo
      FinPara
      Imprimir, "Monto total a pagar es", tot
Fin
```

HastaQue (variable) sea (variable) [Ciclos repetitivos inexactos]

Ejemplo: Ingresar los sueldos de los empleados de una empresa hasta que el empleado sea igual a 0. Calcular el total de sueldos a pagar.

En seudocódigo:

```
Comienzo
      Ingresar, "Ingrese empleado"
      Leer, emp
      Hacer mientras emp <> 0
            Ingresar, "Ingrese sueldo"
            Leer, sdo
            tot = tot + sdo
            Ingresar, "Ingresar empleado"
            Leer, emp
      Repetir
      Imprimir, "Monto total a pagar es", tot
Fin
```

9. ¿Qué es una función[1]? Compárela con un procedimiento[2]

Una función es una sección de un programa que calcula un valor de manera independiente al resto del programa. Una función tiene tres componentes importantes:

- Los parámetros, que son los valores que recibe la función como entrada;
- El código de la función, que son las operaciones que hace la función;
- El resultado (o valor de retorno), que es el valor final que entrega la función.

En esencia, una función es un mini programa. Sus tres componentes son análogos a la entrada, el proceso y la salida de un programa.

---

[1] Se hará la siguente distinción entre software, programa y aplicación:
Software: los programas y otra información operativa que utiliza una computadora. Es un término que lo abarca todo y que a menudo se usa en contraste con el hardware (las partes tangibles de una computadora).
Programa: los programas que se ejecutan en segundo plano y que utiliza el sistema operativo. No fueron desarrollados para el usuario final.
Aplicación: los programas que tienen una interfaz gráfica de usuario y que utiliza el usuario final.

[2] Los diferentes pasos (acciones) de un algoritmo se expresan en los programas como instrucciones, sentencias o proposiciones. El término instrucción se suele referir a los lenguajes máquina y bajo nivel, reservando la sentencia o proposición para los lenguajes de alto nivel.

Por otro lado, una función puede realizar acciones sin entregar necesariamente un resultado. Por ejemplo, si un programa necesita imprimir cierta información muchas veces, conviene encapsular esta acción en una función que haga los print. En este caso, cada llamada a la función imprimir_datos muestra los datos en la pantalla, pero no entrega un resultado. Este tipo de funciones son conocidas en programación como procedimientos o subrutinas.

Técnicamente, todas las funciones retornan valores. En el caso de las funciones que no tienen una sentencia return, el valor de retorno siempre es None. Pero como la llamada a la función no aparece en una asignación, el valor se pierde, y no tiene ningún efecto en el programa principal.