U1: Introducción a los Lenguajes y Paradigmas de Programación

Se puede definir a un paradigma como:
Una actividad con un objetivo determinado.
*Un modelo o ejemplo a seguir por una comunidad científica.
Una filosofía de construcción de software.
Un proceso sin guías por el cual se alcanzan resultados científicos.

Los paradigmas lógico y funcional:
*Pertenecen al paradigma declarativo.
Pertenecen al paradigma Imperativo.
No tienen nada en común.
Son exactamente lo mismo.

Un paradigma es:
Un conjunto de opciones.
Un entorno de desarrollo.
*Un modelo.
Un caso de éxito.

Un Paradigma de Programación:
Define cómo deben interpretarse los resultados de la investigación científica.
Determina el tipo de interrogantes a formular para hallar respuestas en relación al objetivo.
*Es un enfoque particular o filosofía para la construcción del software.
Determina una manera especial de entender el mundo, explicarlo y manipularlo.

Señale cuál de los siguientes no es un paradigma:
OOP (Object Oriented Programming).
Lógico.
Procedural.
*SOA (Service Oriented Architecture).

Paradigma imperativo:
Objetos + Mensajes = Programa.
Funciones + Control = programa.
*Algoritmo + Estructura de Datos = programa.
Lógica + Control = programa.

En el paradigma imperativo se realizan:
*Asignaciones destructivas.
Asignaciones no destructivas.
Asignaciones que no provocan efecto de lado.
Asignaciones que no afectan a los datos

En el paradigma imperativo un programa indica:
El qué del dominio o situación.
*El cómo se va procesando paso a paso la información.
Una no secuencia del paso a paso del procesamiento de la información.
La descripción de la situación.

En un programa del Paradigma Imperativo:
Su codificación consiste en describir las propiedades de la solución buscada.
*La solución se describe detallando una secuencia ordenada de tareas a realizar.
Está constituido por hechos, reglas, restricciones, ecuaciones, transformaciones.
Es el único paradigma que utiliza variables globales.

Cuando una solución tiende a lo procedimental:
*Decimos cómo resolver un problema.
Expresamos características del problema.
Tenemos un menor control sobre el algoritmo.
Se requiere de algún mecanismo externo para resolver el algoritmo.

La Programación estructurada combina dos ideas:
Declaraciones y Premisas.
Instrucciones y Mensajes.
*Flujo de control e invariantes.
Premisas y Mensajes.

En la transparencia referencial:
Se actualizan los estados de información.
Se realizan asignaciones destructivas.
*No se actualizan los estados de información.
Se produce el efecto de lado.

Cuando una solución tiende a lo declarativo:
Tenemos secuencia.
Decimos cómo resolver el problema.
*Expresamos características del problema.
Tenemos un mayor control sobre el algoritmo.

Bajo el paradigma declarativo, un programa expresa:
Cómo se soluciona paso a paso un problema.
*Qué es la solución de un problema.
Quién soluciona el problema.
Cuándo se soluciona el problema.

El Paradigma lógico:
Se caracteriza por encapsular datos y procedimientos.
*Puede resumirse en la expresión: Axiomas + reglas de deducción = programa.
El programador tiene la responsabilidad de un adecuado flujo de control.
Se destaca por su eficiencia y uso masivo en aplicaciones reales.

Bajo el paradigma de objetos, un programa puede contener sentencias imperativas:
*Sí, cuando la implementación de ese lenguaje no es totalmente pura.
No, nunca es posible incluir sentencias imperativas.
Sí, pero sólo si ese programa está compuesto únicamente de sentencias imperativas.
Sí, pero sólo en casos de implementación de polimorfismo.

En el Paradigma Orientado a Objetos:
*El sistema se descompone en objetos con responsabilidades bien especificadas.
Es el único paradigma cuyos lenguajes no utilizan variables globales.
Una ventaja importante de este paradigma es la ausencia de efectos colaterales.
Todas sus sentencias y datos se encapsulan en módulos.

La programación concurrente y el paralelismo en el hardware:
Son exactamente lo mismo, es el mismo concepto.
No son el mismo concepto, pero no puede darse uno sin el otro.
Son uno consecuencia del otro.
*Son dos conceptos absolutamente independientes.

La sintaxis de un lenguaje de programación está conformada por:
*Reglas que permiten la formación de sentencias válidas de en un lenguaje.
Reglas que brindan el significado de una sentencia o instrucción del lenguaje.
Reglas que especifican los elementos léxicos que forman programas en el lenguaje que se está definiendo.
Reglas que facilitan la formación y significado de sentencias válidas en un lenguaje.

¿Cuál es el concepto más apropiado de semántica de un lenguaje?:
Definición formal de la sintaxis de un lenguaje de programación.
Conjunto de reglas que gobiernan la construcción o formación de sentencias (instrucciones) válidas en un lenguaje.
*Es el conjunto de reglas que proporcionan el significado de una sentencia o instrucción del lenguaje.
Es el aspecto que ofrece el programa.

El lenguaje C puede considerarse como un lenguaje de programación asociado al paradigma:
Funcional.
Lógico.
*Imperativo.
Concurrente.

El lenguaje Pascal puede considerarse con un lenguaje de programación asociado al paradigma:
Lógico.
Funcional.
*Imperativo.
Concurrente.

El lenguaje Lisp puede considerarse con un lenguaje de programación asociado al paradigma:
Lógico.
Imperativo.
*Funcional.
Concurrente

Un programa se dice que es fiable si:
Realiza sus acciones mediante el uso de subrutinas.
*Realiza sus especificaciones en cualquier condición.
Realiza sus especificaciones con pocas estructuras que permiten la construcción de sistemas complejos.
Realiza sus especificaciones con pocas estructuras, mediante el uso de subrutinas que permiten la construcción de sistemas complejos.

Señale la afirmación correcta respecto a los lenguajes de programación híbridos:
Smalltalk es un ejemplo de un lenguaje de programación híbrido.
Son muy estrictos en cuanto a su implementación.
Mantienen las características propias del paradigma al cual está asociado.
*Agregan características propias de otros paradigmas.

Un programa es:
El sistema operativo.
*Conjunto de instrucciones ordenadas correctamente que permiten realizar una tarea o trabajo específico.
Una serie de reglas determinadas por el lenguaje a seguir para solucionar un problema.
Es la herramienta que permite expresar la solución a un problema.

U2: Elementos Constitutivos de Lenguajes y Paradigmas

Abstracción es:
*Extraer las características esenciales del objeto.
Extraer las características no esenciales y esenciales del objeto.
Extraer las características accidentales de un objeto.
Extraer las características no esenciales del objeto.

Cada módulo conoce:
El funcionamiento interno de los demás módulos.
El funcionamiento interno de los demás módulos y su interfaz.
*Sólo conoce la interfaz de los que interactúa.
El funcionamiento externo e interno de los demás módulos.

La modularización es una estrategia de la programación que persigue:
*La organización y distribución de la funcionalidad de un sistema complejo en parte más pequeñas de software.
La distribución en la invocación que desde un módulo se efectúa a otro módulo.
La organización y separación del conocimiento sobre la definición del problema, logrando una separación entre la lógica y el control.
La organización del programa en un conjunto de declaraciones.

Característica de un módulo:
Alto grado de acoplamiento.
*Poco acoplamiento.
Baja Cohesión.
Tamaño relativamente grande.

¿Cuál de las siguientes afirmaciones no está asociada con los conceptos de modularidad y encapsulamiento?:
Cada módulo conoce de los demás módulos con los que interactúa, sólo la interfaz.
*Cada módulo conoce el detalle del funcionamiento interno de los demás módulos con los que interactúa.
Ante la modificación de una funcionalidad en particular del sistema, en la medida que su implementación esté encapsulada en un módulo, el impacto que produce su cambio no afectará a los otros módulos que interactúan con él.
Cada módulo sólo conoce de los demás módulos con los que interactúa, la forma en que debe enviarle información adicional en forma de parámetros y cómo va a recibir las respuestas.

El concepto de transparencia referencial consiste en:
*En que el valor de una expresión depende únicamente del valor de sus componentes, es decir que si se evalúa el mismo bloque de software con los mismos parámetros se obtiene el mismo resultado.
En que el valor de una expresión que no depende del valor de sus componentes, es decir que si se evalúa el mismo bloque de software con los mismos parámetros se obtiene el mismo resultado.
En que el valor de una expresión que no depende del valor de sus componentes, es decir que si se evalúa el mismo bloque de software con diferentes parámetros se obtiene el mismo resultado.
En que el valor de una expresión que depende del valor de sus componentes, es decir que si se evalúa el mismo bloque de software con diferentes parámetros se obtiene el mismo resultado.

Orden Superior significa:
Realizar una evaluación diferida.
*Que una función pueda ser utilizada como parámetro y devuelta como resultado.
Que una función pueda ser utilizada como parámetro y no devuelta como resultado.
Que una función pueda ser devuelta como resultado y no como parámetro.

Orden superior en un programa significa:
*Los programas pueden tener como argumento Programas y producir como resultado otros Programas.
Los programas cuentan con argumentos de entrada que serán procesados en el void main().
Los programas pueden realizar llamadas a funciones recursivas.
Los programas serán evaluados en tiempo de ejecución por las máquinas virtuales.

En un programa declarativo las declaraciones pueden ser:
*Proposiciones.
Motores.
Correcciones.
Modificaciones.

Los tipos primitivos son:
*Valores atómicos.
Aquellos que pueden ser descompuestos en otros valores.
Valores atómicos y valores que se descomponen en otros valores.
Conforman una estructura de datos.

¿Cuál de las siguientes afirmaciones no está asociada con el concepto de estructura de datos? :
Es conjunto de valores.
*Es un valor atómico.
Se pueden procesar en su conjunto como una unidad.
Se pueden descomponer en partes y tratar cada una en forma independiente.

El tipo de dato al que pertenece una entidad determina:
La forma en que puede convertirse a otro tipo.
La manera en que sus componentes se representan en sistema binario.
*La operatoria que se puede realizar con ella.
Las operaciones matemáticas posibles.

Un tipo de dato abstracto es:
Un tipo sin definición.
Un tipo declarado como abstract.
*Un conjunto de valores y de operaciones asociadas a ellos.
Una nómina de tipos.

La transferencia referencial:
*Depende únicamente del valor de sus componentes.
Siempre que se evalúa el mismo bloque se obtiene otro resultado.
Se cambian en forma constante los valores de los parámetros.
Se produce el efecto de lado.

La asignación destructiva:
No produce efecto de lado.
*Se obtiene un resultado distinto al efectuarla.
No se cambian en forma constante los valores de los parámetros.
Es la forma menos usual de provocar efecto de lado.

Transparencia referencial:
Es una característica de los lenguajes OO puros.
*Consiste en que el valor de una expresión depende únicamente del valor de sus componentes.
Es un prerrequisito de la abstracción y encapsulamiento.
Es la consecuencia más importante del efecto colateral.

Unificación es un mecanismo por la cual:
*Una variable que no tiene valor, asume un valor.
Una variable que tiene valor, asume un valor.
Una variable a la que se le asigna un valor.
Una constante que asume valores a través del tiempo.

En una evaluación ansiosa los argumentos son:
Evaluados después de invocar al bloque de software.
Evaluados durante la invocación del bloque de software.
Se evalúan siendo responsabilidad del bloque de software invocado.
*Evaluados antes de invocar el bloque de software.

Los parámetros que se utilizan en la invocación de un bloque de software cualquiera:
Deben ser siempre evaluados antes de resolver el bloque.
*Pueden ser evaluados en diferentes momentos de acuerdo al modo de evaluación.
Deben ser evaluados antes o después según la caracterización del parámetro.
Deben ser evaluados antes de la invocación del bloque.

Redefinir un método en clases que se hereda de una clase base se llama:
Generalización.
*Especialización.
Extensibilidad.
Escalabilidad.

En la recursividad:
Un bloque de código recursivo no puede invocarse a sí mismo.
Un bloque de código recursivo no puede invocar a otros bloques de código.
*Un bloque de código recursivo se invocará a sí mismo.
Un bloque de código recursivo solo invocará a otros bloques de código.

Un lenguaje fuertemente tipado se caracteriza porque:
Toda variable o parámetro pueden en ocasiones ser definidos por un tipo de datos en particular que se mantiene sin cambios durante la ejecución del programa.
Toda variable o parámetro no deben ser definidos por un tipo de datos en particular que se mantiene sin cambios durante la ejecución del programa.
*Toda variable o parámetro deben ser definidos por un tipo de datos en particular que se mantiene sin cambios durante la ejecución del programa.
Toda variable o parámetro deben ser definidos por un tipo de datos en particular que permite la posibilidad de cambios durante la ejecución del programa.

La coerción consiste:
En la conversión explícita o automática de un tipo a otro.
En la conversión explicita, el programador se encarga de hacer las conversiones directamente en código.
En la conversión implícita, el programador se encarga de hacer las conversiones directamente en código.
* En la conversión implícita o automática de un tipo a otro.

En un lenguaje fuertemente tipado:
*Toda variable y parámetro deben ser definidos de un tipo de dato en particular.
Pueden asumir valores y tipos de datos diferentes durante la ejecución del programa.
El compilador se encargará de determinar qué tipo de dato es una variable.
No existen los lenguajes "Fuertemente tipados".

¿Cuál de los siguientes lenguajes usa tipificación estática de datos? :
Smalltalk.
*C.
Python.
Lisp.

¿Cuál de los siguientes lenguajes usa tipificación estática?:
Smalltalk.
*Java.
xBase.
Perl.

¿Cuál de los siguientes lenguajes usa tipificación dinámica?:
Java.
*Smalltalk.
C.
C++.

El conocimiento de la clasificación de los valores en tipos permite al programador:
Evitar errores de lógica.
*Evitar la ejecución de operaciones sin sentido.
Utilizar un esquema de tipificación estática eficiente.
Definir criterios de encapsulamiento lo más adecuados posible.

El objetivo básico del polimorfismo es:
Lograr intercambio de datos entre diferentes lenguajes.
Lograr intercambio de datos entre diferentes paradigmas.
Construir software basado en múltiples paradigmas.
*Construir piezas de software genéricas que trabajen indistintamente con diferentes tipos de entidades.

El polimorfismo está asociado a:
Los datos de los procesos.
Los atributos de los objetos.
*Los procesos o comportamientos.
La estructura de los programas

El polimorfismo de sobrecarga se da:
*En clases completamente independientes.
Se da en funciones con funcionalidades diferentes.
Con funciones de distinto nombre.
En clases no independientes.

El Polimorfismo de subtipado consiste en:
*La posibilidad de llamar un método de objeto, sin tener que conocer su tipo intrínseco.
La posibilidad de definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes (nombre y/o tipo).
La posibilidad de definir varias funciones utilizando nombres diferentes, usando parámetros diferentes (nombre y/o tipo).
La posibilidad de definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes (nombre).

U3: Paradigma Imperativo

En el paradigma imperativo, un programa es:
*Un conjunto de acciones precisas a realizar.
Un bloque de operaciones matemáticas.
Un conjunto de declaraciones.
Un conjunto de expresiones regulares.

La programación estructurada es:
*Una técnica en el cual la estructura de un programa se realiza mediante el uso de estructuras lógicas de control: Secuencia, Selección, Iteración.
Una técnica en el cual la estructura de un programa se realiza mediante el uso de estructuras lógicas de control: Secuencia e Iteración.
Una técnica en el cual la estructura de un programa se realiza sin el uso de estructuras lógicas de control.
Una técnica en el cual la estructura de un programa se realiza exclusivamente mediante el uso de estructuras lógicas de control denominadas Iteración.

Respecto a la programación estructurada:
*Toda función computable puede ser implementada mediante secuencia, selección o iteración.
En este estilo de programación se suprimen posibles efectos colaterales.
Una instrucción "go to" permite pasar el control a cualquier otra parte del programa.
En este estilo de programación es posible la coexistencia de varios flujos de control simultáneos.

Respecto a la programación estructurada:
Proporciona un mejor nivel de abstracción que la POO.
La codificación se organiza en rutinas o procedimientos almacenados en clases.
*La programación estructurada tiende a ser orientada a la acción.
La unidad fundamental de esta programación son las entidades.

¿Cuál de las siguientes afirmaciones NO es correcta en el paradigma estructurado?:
Toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras de control.
Las estructuras de control se pueden agrupar en: secuenciales, selectivas e iterativas.
*Un buen programa estructurado sólo necesita de una sola instrucción (go to) para modificar la secuencia de ejecución de las instrucciones mediante la transferencia incondicional de su control.
En las estructuras secuenciales las instrucciones se van ejecutando una tras otra.

El paradigma estructurado está fuertemente basado en:
*Variables para conservar estado de los datos de un programa.
Objetos para conservar el estado de los datos de un programa.
Funciones para conservar el estado de los datos de un programa.
Declaraciones para conservar el estado de los datos de un programa.

La abstracción en la programación estructurada:
Es a nivel de datos y su comportamiento asociado originando Tipos de Datos Abstractos.
*Es una abstracción de comportamiento.
Es realizada a nivel de mensajes.
Es realizada por medio de funciones recursivas.

En la programación estructurada, la representación gráfica se realiza a través de:
Diagrama de transición de estados.
Diagramas de bloques.
* Diagramas de Flujo o Flow chart.
Diagrama de entidades.

El teorema del programa estructurado establece:
La combinación de dos estructuras lógicas secuencia y selección.
La combinación de tres estructuras lógicas secuencia, interacción y selección.

*La combinación de tres estructuras lógicas secuencia, iteración y selección.
La combinación de dos estructuras lógicas secuencia e iteración.

Una estructura condicional simple indica:
*Una condición y un bloque de acciones a realizar si la misma arroja resultado verdadero.
Una acción y un bloque de condiciones verdaderas.
Una condición y un bloque de acciones a realizar tanto para el caso en que la misma arroja resultado falso como verdadero.
Una condición sin acciones.

El ciclo while:
Permite repetir un grupo de acciones mientras la condición del ciclo sea falsa.
*Permite repetir un grupo de acciones mientras la condición del ciclo sea verdadera.
Permite repetir un grupo de acciones hasta que la condición del ciclo sea verdadera.
Permite repetir un grupo de acciones mientras la condición del ciclo y la variable de control de ciclo sean verdaderas.

El ciclo for:
Permite repetir un grupo de acciones mientras la variable de control de ciclo sea verdadera.
Permite repetir un grupo de acciones hasta que la variable de control de ciclo sea falsa.
Permite repetir un grupo de acciones mientras la condición del ciclo y la variable de control de ciclo sean verdaderas.
*Permite repetir un grupo de acciones mientras la condición del ciclo sea verdadera.

Una función se puede conceptualizar como:
Un subproceso que puede recibir o no parámetros de entrada o de entrada/salida y realiza un conjunto de acciones retornando más de un resultado.
*Un subproceso que puede recibir o no parámetros de entrada o de entrada/salida y realiza un conjunto de acciones retornando un único resultado.
Un subproceso que nunca recibe parámetros y realiza un conjunto de acciones retornando más de un resultado.
Un subproceso que puede recibir o no parámetros de salida y realiza un conjunto de acciones retornando un único resultado.

Las declaraciones de subrutinas generalmente son especificadas por:
Un nombre único en el ámbito, un tipo de dato de retorno, una lista de parámetros (obligatorio) y conjunto de órdenes que debe ejecutar la rutina.
Un nombre único en el ámbito, una lista de parámetros (obligatorio) y conjunto de órdenes que debe ejecutar la rutina.
Un nombre único en el ámbito, un tipo de dato de retorno y conjunto de órdenes que debe ejecutar la rutina.
*Un nombre único en el ámbito, un tipo de dato de retorno, una lista de parámetros (opcional) y conjunto de órdenes que debe ejecutar la rutina.

Un procedimiento:
No recibe parámetros.
*No retorna valores.
Retorna valores.
Retorna un único resultado.

¿Cuál de las siguientes afirmaciones NO es una ventaja de las subrutinas en la programación estructurada?:
*Disminuye la mantenibilidad y extensibilidad del programa.
Aumenta la legibilidad del código de un programa.
Permite la reutilización del código.
Descomposición de problemas complejos en problemas más simples.

¿Cuál de las siguientes afirmaciones NO es correcta respecto a los procedimientos y funciones en la programación imperativa?:
*Los procedimientos son subrutinas que pueden recibir o no algún parámetro de entrada/salida, realizan algún proceso y siempre retornan algún valor.
Las funciones son subprocesos que pueden recibir o no algún parámetro y retornan siempre un único valor.
En lenguaje C sólo existen las funciones y los procedimientos se programan como funciones que no retornan ningún valor.
En el lenguaje Pascal se distingue entre la subrutina procedimiento y la subrutina función.

Una función:
No recibe parámetros.
*Retorna valores.
No retorna un único resultado.
Puede o no retornar un resultado.

Una subrutina permite:
*Reducción de código duplicado.
El no aumento de la legibilidad del código.
No descompone un problema complejo.
No aumenta la extensibilidad.

¿Cuál de las siguientes afirmaciones es correcta? :
Un procedimiento es un subproceso que puede recibir o no parámetros de entrada o de entrada / salida y realiza un conjunto de acciones retornando un único resultado.
*Un procedimiento es un subproceso que puede recibir o no parámetros de entrada o de entrada / salida y realiza un conjunto de acciones sin retornar valores.
Una función es un subproceso que puede recibir o no parámetros de entrada o de entrada / salida y realiza un conjunto de acciones sin retornar valores.
Todas son correctas

Una subrutina es:
Un programa que llama a otro.
Un bloque que se repite.
*Una porción de código que realiza una tarea específica.
Una porción de código que se ejecuta si se cumple una condición.

Respecto a procedimientos y funciones:
Se organizan en clases que implementan conceptos.
*Permiten la reutilización de código en múltiples programas.
Las variables globales definidas en su cuerpo son visibles en todo el programa.
Únicamente los procedimientos pueden recibir diversos tipos de parámetros de entrada.

Respecto al concepto de subrutinas:
Se las considera procedimientos cuando reciben diversos parámetros de entrada.
*Se las considera procedimientos si no retornan valores.
Se las considera procedimientos si contienen estructuras de control.
Se las considera procedimientos si invocan otras subrutinas.

Una función es:
Una subrutina que recibe parámetros.
Una subrutina que no recibe parámetros.
*Una subrutina que siempre retorna un valor.
Una subrutina que no tiene acciones.

Las variables locales son aquellas que:
Se declaran y son accesibles en todo momento.
Se declaran y son accesibles en un contexto determinado. Al finalizar dicho contexto la variable mantiene su valor y puede ser usada.
*Se declaran y son accesibles en un contexto determinado. Al finalizar dicho contexto la variable se destruye perdiendo su valor y toda posibilidad de ser utilizada.
Se declaran y son accesibles en un contexto determinado. Al finalizar dicho contexto la variable no se destruye y puede ser usada.

¿Cuál de las siguientes afirmaciones no es correcta sobre la programación estructurada?:
* El ámbito de una variable local son todas las funciones que puede tener un programa.
Las variables locales se declaran y se pueden utilizar sólo dentro del contexto en el cual son declaradas.
Las variables globales son accesibles desde cualquier función del programa en el cual se las declara.
Las variables son el espacio de almacenamiento en memoria donde los programas imperativos guardan valores que utilizan en los procesos que realizan.

Características comunes en distintas implementaciones de programación imperativa:
*Variables globales, locales y estructuras de control.
Variables globales y condicionales.
Procedimientos y declaratividad.
Mensajes y funciones.

Las variables globales:
*Son accesibles desde cualquier parte del programa.
No son accesibles desde cualquier parte del programa.
Almacenan un valor no perdurable.
Son accesibles en un contexto determinado.

Señale lo que es correcto respecto a la función printf() en el lenguaje C:
*Puede recibir como parámetro una cadena que represente el mensaje a mostrar y un conjunto de variables que contienen los valores que se desea mostrar por pantalla.
Solo puede recibir un único parámetro que es un valor numérico que se desea mostrar por pantalla.
Permite leer por teclado un valor.
Solo puede recibir un conjunto de variables del mismo tipo para ser mostradas por pantalla.

U4: Paradigma de Programación con Orientación a Objetos

El principio de encapsulamiento:
*Permite aumentar la cohesión de los componentes del sistema.
No permite aumentar la cohesión de los componentes del sistema.
Permite aumentar el acoplamiento de los componentes del sistema.
Permite aumentar la abstracción de los componentes del sistema.

El encapsulamiento:
Protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ella.
Permite subdividir una aplicación en partes más pequeñas
*Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción.
Denota las características esenciales de un objeto, donde se capturan sus comportamientos.

El principio de ocultación:
*Protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ella.
Permite subdividir una aplicación en partes más pequeñas.
Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción.
Denota las características esenciales de un objeto, donde se capturan sus comportamientos.

Dos objetos son polimórficos respecto de un conjunto de mensajes si:
*Ambos pueden responder de manera semánticamente equivalente (aún si su implementación es distinta).
Ninguno puede responder.
Ambos pueden responder de manera diferente mediante una lista de colaboradores externos.
Ninguno puede responder a menos que se determinen colaboradores internos.

Sobre el polimorfismo:
Usar polimorfismo implica trabajar sobre una estructura hereditaria de clases.
Es un recurso que todos los lenguajes POO proveen exclusivamente en tiempo de ejecución.
*La invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado.
El polimorfismo es posible porque una referencia a objeto puede instanciarse con cualquier objeto.

Encapsulamiento significa:
*Reunir a elementos pertenecientes a una misma entidad.
Reunir a elementos de distinto nivel de abstracción.
Extraer las características esenciales de un objeto.
Que cada tipo de objeto expone una interfaz.

Cohesión:
*Mide las responsabilidades asignadas a cada objeto.
Mide las relaciones entre los objetos.
No mide las responsabilidades asignadas a cada objeto.
Es lo que el modelo orientado a objetos busca disminuir.

Especifique lo que NO es una característica deseable en la orientación a objetos:
Abstracción.
Recolección de basura.
Principio de ocultación.
*Baja cohesión y alto acoplamiento.

La abstracción es:
*El proceso por el cual se extraen las características y comportamientos esenciales de un objeto.
El proceso por el cual se divide un objeto en componentes más pequeños.
El proceso por el cual se encapsula el comportamiento de un objeto.
El proceso por el cual se modela la interoperatividad entre los objetos.

Sobre la abstracción:
*Denota las características esenciales de un objeto, donde se capturan sus comportamientos.
Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad.
Así se denomina a la propiedad que permite subdividir una aplicación en partes más pequeñas.
Cada objeto está aislado del exterior, es un módulo natural.
Protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a hacerlo.

En la Herencia:
Ninguna de las anteriores afirmaciones es correcta.
Los objetos heredan sólo el comportamiento de las clases a la que pertenecen.
Los objetos heredan sólo las propiedades de las clases a la que pertenecen.
*Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.

Sobre la Herencia:
*La herencia permite definir/crear objetos como tipos especializados de objetos preexistentes.
Todos los lenguajes POO implementan herencia múltiple.
Los lenguajes que implementan herencia requieren de chequeo estático de tipos de datos.
Los lenguajes que implementan herencia necesariamente son débilmente tipados.

Un objeto dentro del Paradigma de objetos debe poseer:
Abstracción, Cohesión, Identidad y Capacidad de Inspección.
*Abstracción, Comportamiento, Identidad y Capacidad de Inspección.
Abstracción, Comportamiento, Acoplamiento y Capacidad de Inspección.
Principio de Ocultación, Comportamiento, Acoplamiento y Capacidad de Inspección.

El comportamiento de un objeto:
Estará definido por las funciones declaradas en la clase.
*Estará definido por el conjunto de mensajes que el objeto pueda responder.
Estará definido por los atributos de la clase.
Estará definido por los procedimientos declarados en la clase.

El comportamiento de un objeto es:
La forma en que se representa en memoria.
El conjunto de datos que puede aceptar y transformar.
El conjunto de mensajes que puede mandar a otros objetos.
*El conjunto de mensajes que el objeto puede responder.

En cada acción de envío de mensaje intervienen:
*Un emisor, receptor, selector, eventualmente parámetros (que son objetos) y eventualmente un resultado (que es otro objeto).
Un emisor, receptor y eventualmente parámetros (que son objetos) y eventualmente un resultado (que es otro objeto).
Un emisor, selector, eventualmente parámetros (que son objetos) y eventualmente un resultado (que es otro objeto).
Un emisor, receptor, selector y eventualmente parámetros (que son objetos).

En Smalltalk un selector es:
Las sentencias de un método.
Un mensaje unario.
*Nombre que identifica el mensaje.
Un bloque.

En Smalltalk, el conjunto de variables que va a contener un objeto se llama:
Parámetro.
Método.
Colaborador externo.
*Colaborador Interno.

¿Qué es un método? :
Es una abstracción de la realidad y siempre tiene asociados variables temporales o locales.
Es una subrutina la cual se puede descomponer a su vez en otras subrutinas.
Conjunto de instrucciones que se ejecutan a la vez.
*Conjunto de colaboraciones que un objeto tendrá con otros, para responder un objeto.

La programación en Smalltalk consiste en:
Creación de clases y especificación de la secuencia de mensajes entre objetos.
*Creación de clases, instancias y especificación de la secuencia de mensajes entre objetos.
Creación de instancias y especificación de la secuencia de mensajes entre objetos.
Creación de clases e instancias.

El operador (:=) en Smalltalk representa:
Operador de concatenación.
Operador de conversión de tipos.
Operador lógico.
*Operador de ligadura o asignación.

Un Workspace permite:
Observar la estructura interna de un objeto.
Editar la biblioteca de clases del ambiente.
*Evaluar expresiones.
Configurar expresiones.

Los literales hacen:
*Referencia siempre al mismo objeto.
Referencia siempre a distintos objetos.
Referencia siempre a distintos objetos en un instante de tiempo.
No referencia siempre al mismo objeto.

Especifique lo que no es un mensaje para ningún objeto en Smalltalk :
+.
factorial.
>=.
*:=.

Pool Dictionaries son:
Métodos de clases.
*Variables contenedoras.
Variables de clase.
Métodos de instancias.

Cuál es el resultado de la siguiente expresión en Smalltalk 4 + 5 * 3 - 3 factorial:
13.
true.
*21.
-27.

En SmallTalk un objeto puede enviarse mensajes a sí mismo:
No.
*Sí, utilizando la palabra clave self.
Sí, anteponiéndose como colaborador externo.
Sí, poniendo su tipo antes del mensaje.
*TE*Implementación del POO en Smalltalk
*ST*Sintaxis y Semántica
*RB*Referencias Bibliográficas no utilizadas

La regla de inferencia que emplea la programación lógica:
*Principio de Resolución
Cláusulas de Horn
Principio de Colmenauer
Ninguna correcta

Los enunciados más básicos del LPO son:
Las cláusulas de Horn
Las constantes individuales
*Los enunciados atómicos
Los símbolos

Las constantes individuales hacen referencia a:
Varios objetos
*Exactamente a un objeto particular
A ningún objeto
Exactamente a dos objetos

La aridad es:
*Un número que indica la cantidad de constantes individuales
Un literal que indica la cantidad de símbolos de predicado
Un número que indica la cantidad de enunciados atómicos
Una constante

Los símbolos de predicado son utilizados para denotar:
Una propiedad
Una relación entre objetos
*Una propiedad y una relación entre objetos
Ninguna de las anteriores

En Prolog para definir las relaciones se utiliza:
Las reglas de Colmenauer
*Las cláusulas de Horn
Las reglas de inferencia
Las reglas de Robinson

Los elementos de un programa en Prolog son:
Base de conocimiento
Base de conocimiento y motor de inferencia
Base de conocimiento, hechos y reglas
*Base de conocimiento, motor de inferencia y Resolución
Base de conocimiento, Motor de inferencia y reglas

Pasos a seguir para escribir un programa en Prolog:
Declarar hechos
Declarar hechos y Base de Conocimiento
Declarar reglas y Base de Conocimiento
*Declarar hechos, reglas y hacer preguntas

La estructura de un programa en Prolog es:
*Lógica + control
Lógica + mensaje
Lógica + función + control

La unificación es un mecanismo para:
Que las variables tomen valores
*Que las variables lógicas tomen valores
Que la variable no esté ligada
Ninguna de las anteriores

El principio de resolución de Prolog se basa en:
La lógica de segundo orden
*La lógica de primer orden
El principio de comprobación de teoremas
Un proceso sin guías por el cual se alcanzan resultados científicos

Indique cuál de las siguientes declaraciones de variables es correcta en SWI-Prolog:
*_edad
$nombre
%telefono
5Direccion

El símbolo :- es el símbolo de:
Negación
Conjunción
Disyunción
*Implicación

En las cláusulas de Horn, el consecuente es:
*Lo que se quiere probar
Una condición que realiza la comprobación
Ambas respuestas son correctas
Ninguna respuesta es correcta

¿Cuál de los siguientes NO es un elemento de un programa Prolog?:
Base de Conocimiento
Motor de Inferencia
Resolución
*Compilador de predicados

La implementación de la Regla de Inferencia en Prolog se basa en:
Backtracking y resolución
*Unificación y backtracking
Unificación y resolución
Backtracking y corte

La Unificación en Prolog:
Es una comprobación de igualdad entre variables
*Es el mecanismo mediante el cual las v ariables lógicas toman valor en Prolog
Es una simple asignación de valores a variables
Es un mecanismo de resolución de valores de variables

El corte en Prolog permite:
Evitar la unificación de variables
*Controlar el backtracking
Optimizar el motor de inferencia
Ninguna de las anteriores

Si quisiera simular un ciclo en Prolog:
*Debería utilizar recursividad
Debería crear un predicado especificando la palabra clave repeat
Debería usar corte y backtracking
No es posible simular un ciclo en Prolog

Cuando uso listas en Prolog:
Puedo acceder a cualquier elemento de la lista usando get_element (index)
Puedo recorrerla usando un ciclo
*Puedo especificar los términos de la lista en forma independiente como cabecera y cola
Ninguna de las anteriores

Los símbolos de predicado son:
*Expresiones que combinadas con nombres, forman enunciados atómicos
Enunciados formados por un prefijo que son los argumentos
Funciones que transforman los argumentos de objeto (constantes individuales) en valores verdaderos o falsos
Son símbolos (nombres) que se usan para referir a algún objeto individual fijo

En Prolog se utiliza un determinado tipo de reglas para definir relaciones llamadas:
Reglas de inferencia lógicas
*Cláusulas de Horn
Reglas Deterministas
Cláusulas de Unificación

Un programa en Prolog está conformado por una serie de elementos:

Hechos, Reglas y Motor de Inferencia
Base de Conocimiento, Hechos y Reglas
Base de Conocimiento y Motor de Inferencia
*Base de Conocimiento, Motor de Inferencia y Resolución

Identifique que características NO corresponden a una regla en Prolog:
Una regla consiste en una cabeza y un cuerpo, unidos por el signo ":-"
La cabeza está formada por un único hecho
*Las reglas finalizan con punto y coma (;)
El cuerpo puede ser uno o más hechos separados por una coma (","), que actúa como el "y" lógico

La manera de implementar la Regla de Inferencia en Prolog se basa en los conceptos de:
Backtracking y reglas de inferencia
Unificación y clausulas de Horn
*Unificación y backtracking
Unificación y Hechos

Indicar cuál de la siguiente norma NO corresponde para saber si dos términos unifican:
Una variable siempre unifica con un término, quedando ésta ligada a dicho término
Dos variables siempre unifican entre sí, además, cuando una de ellas se liga a un término, todas las que unifican se ligan a dicho término
Para que dos términos unifiquen, deben tener el mismo functor y la misma aridad
*Si dos términos unifican, ninguna variable queda ligada

Señalar cuál es la característica correcta que identifica al predicado corte:
Se representa por un signo (~)
Tiene la propiedad de provocar el backtracking cuando se produce
Predicado predefinido que recibe argumentos
*Tiene la propiedad de eliminar los puntos de elección del predicado que lo contiene

Señalar qué predicado no se aplica para el manejo de listas en SWI-Prolog:
member(?Elem, ?List)
append(?List1, ?List2, ?List3)
sort(+List, Sorted)
*is sort(+Term)

¿Qué característica no corresponde al lenguaje Prolog?
Lenguaje de programación seminterpretado
Su código fuente se compila a código de byte el cual se interpreta en una máquina virtual
Es multiplataforma
*No es código abierto

Los elementos constitutivos de un programa en Prolog son:
*Hechos, Reglas y Preguntas
Hechos y Reglas
Hechos y Preguntas
Ninguna de las opciones anteriores

Cuál de las afirmaciones es correcta:
La lógica proposicional utiliza proposiciones y nexos entre éstas para expresar acciones
La lógica proposicional utiliza proposiciones y nexos entre éstas para expresar instrucciones
*La lógica proposicional utiliza proposiciones y nexos entre éstas para expresar verdades
La lógica proposicional utiliza proposiciones y nexos entre éstas para expresar estrategias

Cuál de las afirmaciones es incorrecta:
La lógica proposicional utiliza proposiciones y nexos entre éstas para expresar aserciones
*La lógica proposicional utiliza proposiciones y nexos entre éstas para expresar instrucciones
La lógica proposicional utiliza proposiciones y nexos entre éstas para expresar verdades

¿Cuál de las afirmaciones es incorrecta?:
*Un símbolo de predicado denota objetos
Un símbolo de predicado denota relaciones entre objetos
Un símbolo de predicado denota propiedades de los objetos
Un símbolo de predicado es una expresión que combinada con nombres, forman enunciados atómicos

Las cláusulas de Horn representan:
Hechos
Reglas
Objetivos
*Todas las anteriores

¿Cuál de las afirmaciones es incorrecta?:
La unificación representa la igualdad lógica
La unificación permite ligar a las variables
*La unificación es la asignación de múltiples valores en memoria
La unificación es el mecanismo mediante el cual las variables lógicas toman valor en Prolog

Una lista en Prolog es:
Un conjunto de elementos encerrados entre paréntesis y separados por comas
*Un conjunto de elementos encerrados entre corchetes y separados por comas
Un conjunto de elementos encerrados entre corchetes y separados por puntos y comas
Un conjunto de elementos encerrados entre corchetes y separados por puntos

Cuál de las siguientes expresiones es correcta:
X is 8, X = 3+5.
Y is X+5, X = 8.
*X is 8, X is 3+5.
X is 8, X is Y+2.

Cuál es el resultado de evaluar la siguiente expresión X is 7, Y is 7, X>5, Y is X+Y.:
true
*false
No se puede evaluar, está expresada en forma incorrecta

Símbolos de predicado:
Definen una actividad con un objetivo determinado.
*Son utilizados para denotar alguna propiedad o relación entre objetos
Corresponden exactamente a los predicados de la gramática española
En LPO todos los predicados que usemos en un programa deben tener una idéntica aridad

La programación lógica en Prolog
Trabaja con relaciones que tienen sentido de dirección preestablecido
*Se basa en la premisa de que programar con relaciones es más flexible que programar con funciones
Utiliza principalmente un tipo de reglas para definir relaciones, las llamadas cláusulas de Churh
Las cláusulas de Horn permiten crear un lenguaje de primer orden con una sintaxis flexible

Las Cláusulas de Horn:
No tienen un formato definido, pueden adoptar diversas formas
Tienen una forma única, conteniendo antecedente y consecuente
*Tienen tres formas, (Afirmación, Implicación, Negación)
La forma Negación permite negar hechos no verdaderos

Elementos de un programa en Prolog:
Los elementos se distribuyen en cuatro secciones específicas
Ninguna de estas secciones puede faltar
*Base de Conocimiento, Motor de Inferencia, Resolución
Está constituido por hechos y reglas, siempre en ese orden

Las reglas en Prolog y sus características:
Una regla consiste en una cabeza y un cuerpo, unidos por el signo " >"
La cabeza es el conjunto de hechos que deben satisfacerse
*Las reglas se utilizan en Prolog para significar que un hecho depende de otros
El cuerpo debe estar constituido por un único hecho

Las preguntas o consultas en Prolog.
Las preguntas son algunas de las herramientas que tenemos para recuperar información
Deben ser formuladas en la sección Goals
La respuestas False significan que todos los hechos del cuerpo de la regla son falsos
* Prolog es un programa conformado por un conjunto de hechos y reglas que representan el problema que se pretende resolver

La resolución es una regla de inferencia que:
*Permite a la computadora decir qué proposiciones siguen lógicamente a otras proposiciones.
Permite a Prolog resolver alternativamente un problema sin usar las cláusulas de Horn.
Puede implementarse basándose únicamente en el concepto de unificación
Excluyen de esta regla las cláusulas cuyos resultados lleven al absurdo

Respecto a la Unificación, indique qué es falso:
Una variable siempre unifica con un término, quedando ésta ligada a dicho término
*La unificación es equivalente a la asignación de los lenguajes imperativos
Para que dos términos unifiquen, deben tener el mismo functor y la misma aridad
Si dos términos no unifican, ninguna variable queda ligada

Respecto al Backtracking, indique qué es falso:
Cuando se va a ejecutar un objetivo Prolog registra ordenadamente en una pila todos sus puntos de elección
Los puntos de elección (objetivos) apilados son recorridos mientras tengan éxito
Al fallar un objetivo Prolog da marcha atrás (backtracking) recorriendo los objetivos que tuvieron éxito
*El mecanismo de backtracking es automático, el programador no puede controlarlo

Respecto al corte en el backtracking de Prolog, indique qué es falso:
* Sólo se lo usa en ocasiones excepcionales, por introducir indeterminismo en un lenguaje declarativo
Tiene la propiedad de eliminar los puntos de elección del predicado que lo contiene
En la ejecución normal el corte no hace nada
En backtracking con los objetivos recorridos marcha atrás, al llegar al corte el backtracking se detiene repentinamente forzando el fallo del objetivo

Señale cuál no es un campo de aplicación reconocido del Paradigma Lógico:
Sistemas basados en el conocimiento
Inteligencia artificial
*Sistemas comerciales de facturación
Demostración automática de teoremas

Señale lo que no corresponda respecto a las constantes individuales:
Son simplemente símbolos (nombres) que se usan para referir a algún objeto individual fijo
Ningún nombre puede referir a más de un objeto
*Pueden hacer referencia a más de un objeto en particular
Un objeto puede tener más de un nombre

Señale lo que no corresponda respecto a los símbolos de predicados:
Son utilizados para denotar alguna propiedad o relación entre los objetos
Todo símbolo de predicado viene con una aridad fija
*Los símbolos predicados de aridad 1 son los más utilizados para denotar una relación entre más de dos objetos
Son expresiones que combinadas con nombres, forman enunciados atómicos

Prolog, es un lenguaje de programación:
Imperativo
Orientado a objetos
*Declarativo
Estructurado

Señale lo que NO corresponda respecto a un programa en Prolog:
En Prolog es posible representar el conocimiento que se tiene sobre un determinado dominio
El dominio es un conjunto de objetos y el conocimiento se representa por un conjunto de relaciones que describen las propiedades de los objetos y sus interrelaciones
*En Prolog, un programa se basa en el modelo de Von Neumann
En un programa en Prolog, lo fundamental es expresar bien el conocimiento que se tenga sobre el dominio del problema que se esté enfrentando

¿Cuál de los siguientes es un ejemplo de un símbolo lógico en Prolog?:
El símbolo del identificador de una variable individual
El símbolo de una constante
*Símbolos auxiliares de escritura como los corchetes
El conjunto de símbolos de relaciones n-arias

Selecciona lo que NO corresponda respecto a la unificación:
Es el mecanismo mediante el cual las variables lógicas toman valor en Prolog
Cuando una variable no tiene valor se dice que está libre
Si se produce unificación la variable está ligada
*Es el mismo concepto que el de asignación en los lenguajes

Selecciona lo que NO corresponda respecto al corte:
Es un predicado predefinido que no recibe argumentos
Se representa mediante el símbolo: ! (símbolo de admiración de cierre)

Cada una de las alternativas se denomina punto de elección

*Se lo usa frecuentemente para permitir que un programa encuentre todas las soluciones posibles a una pregunta

Tiene la propiedad de eliminar los puntos de elección del predicado que lo contiene

U6: Paradigma Funcional

La Transparencia referencial permite:
El cambio de estado
*Que una expresión tenga siempre el mismo valor
Definir funciones de orden superior
Que una expresión cambie su valor

En los lenguajes funcionales la gestión de la memoria la hace:
El programador
*Se hace implícitamente
El programador en cuanto a asignación y se recupera por el recolector de basura
El programador en cuanto a recuperación de los datos

Las ventajas del paradigma funcional son:
Fácil de integrar con otras aplicaciones
Recomendable para modelar la lógica de negocio
*Administración automática de la memoria
Complejidad en el código

Haskell es un lenguaje apropiado para:
Programas altamente modularizados
Programas altamente modificados
*Programas altamente modificados y mantenidos
Ninguna de las anteriores

Las funciones en Haskell pueden ser:
Argumentos
Argumentos o resultados
*Argumentos o resultados o componentes de estructuras de datos
Resultados

El proceso de currificación permite:
*Reducir el número de paréntesis necesarios para escribir expresiones
Reducir el número de corchetes para escribir expresiones
Ampliar el número de paréntesis para escribir expresiones
Reducir el número de corchetes necesarios para escribir expresiones

La siguiente expresión XS ++ YS devuelve en Haskell:
*La lista resultante de concatenar XS e YS
La suma resultante de XS e YS
La lista de valores obtenidos al aplicar la función XS a YS
Ninguna de las anteriores

El espaciado se basa en que:
Las expresiones están alineadas por filas
*Las expresiones están alineadas por columnas
Las expresiones están alineadas por filas y columnas
Las expresiones no están alineadas

La sentencia Data permite:
*Definir nuevos tipos de datos
Aplicar recursividad
Definir funciones
Declarar tipos predefinidos

La evaluación perezosa:
Evalúa los argumentos de una función antes de conocer si estos serán utilizados
*No evalúa a los argumentos de una función hasta que no se necesita
Permite definir nuevos tipos de datos
Ninguna de las anteriores

Cuál de las afirmaciones es correcta:
*En el paradigma funcional la manera de construir abstracciones es a través de funciones
En el paradigma funcional la manera de construir abstracciones es a través de instrucciones
En el paradigma funcional la manera de construir abstracciones es a través de estructuras
En el paradigma funcional la manera de construir abstracciones es a través de tuplas

Cuál de las afirmaciones es correcta:
El almacenamiento lo asigna el desarrollador y se inicializa implícitamente, y es recuperado automáticamente por un recolector de basura
El almacenamiento lo asigna e inicializa el desarrollador, y es recuperado automáticamente por un recolector de basura
*El almacenamiento se asigna y se inicializa implícitamente, y es recuperado automáticamente por un recolector de basura
El almacenamiento lo asigna, inicializa y recupera el desarrollador

Las tuplas en Haskell son:
Infinitas y contienen elementos de distintos tipos
*Finitas y contienen elementos de distintos tipos
Infinitas y contienen elementos del mismo tipo
Finitas y contienen elementos del mismo tipo

Las listas en Haskell son:
Infinitas y contienen elementos de distintos tipos
Finitas y contienen elementos de distintos tipos
*Infinitas y contienen elementos del mismo tipo
Finitas y contienen elementos del mismo tipo

Los tipos NO básicos en Haskell son:
Enteros, Flotantes, String y Booleanos
Enteros, Flotantes, Caracteres y Booleanos
Enteros, Flotantes, Caracteres, Booleanos y tuplas
Enteros, Flotantes y listas
*Listas, tuplas y funciones

Cuál es el resultado de evaluar la siguiente expresión let x = 2 * 4 in 3 * x + 1 + mod (x) 2:
32
*25
45
13

Cuál es el resultado de evaluar con 2 y 3 la siguiente expresión function x y = if x >= y then x*5/y else y*2/x:
5.0
7.5
*3.0
15

Cuál es el resultado de evaluar con 3 y 2 la siguiente expresión funcion x y = if x >= y then x*5/y else y*2/x:
5.0
*7.5
3.0
15

Cuál es el resultado de evaluar la siguiente expresión [1,3,10] + [2,6,5,7]:
[1, 2, 3, 5, 6, 7, 10]
[1, 3, 10, 2, 6, 5, 7]
*Ninguna no se puede evaluar el operador + en listas

¿Qué es una función de orden superior? :
Es una función que se llama a sí misma hasta una condición de corte especificada
Es una función que retorna sus valores sólo en el campo de los números reales
Es cualquier tipo de función que usamos con expresiones lambda
*Es una función tal que alguno de sus ar gumentos es una función o que devuelve una función como resultado

La evaluación ansiosa en un lenguaje funcional:
*Evalúa todos los argumentos de la función antes de conocer si serán utilizados
No evalúa ningún argumento de la función hasta que se los necesita
No se aplica el concepto de evaluación ansiosa en lenguajes funcionales
No existe ese tipo de evaluación

Para escribir una función en Haskell necesito:
Negación, conjunción y disyunción
*Definición de signatura e implementación del cuerpo de la función
Definición de parámetros de la función
Definición de condición de corte recursivo

Señale la afirmación incorrecta - En la inferencia de tipos:
El programador no está obligado a declar ar el tipo de las expresiones
El compilador contiene un algoritmo que i nfiere el tipo de las expresiones
Si el programador declara el tipo de algu na expresión, el sistema chequea que el tipo declarado coincide con el tipo inferid o.
*El sistema no siempre chequea los tipos, si no coinciden con lo que el programador declaró, se infieren.

Los condicionales en Haskell:
*Deben expresarse siempre como condicional completo (IF..THEN..ELSE)
Pueden expresarse como condicionales simples (IF..THEN)
No es posible anidar condicionales
Sólo se puede evaluar una sola condición, nunca múltiples condiciones

Las expresiones case en Haskell:
Evalúan una condición y comparan con resultados de cadenas
*Evalúan sólo una condición y comparan con resultados numéricos
Evalúan sólo una condición y comparan con cualquier tipo de resultado (numérico o cadena)
Evalúan múltiples condiciones en cada caso

En una función polimórfica:
*El tipo devuelto depende de los argumentos ingresados
El tipo devuelto depende de las variables declaradas con Let o Where
El tipo devuelto depende de un casteo especial
El tipo devuelto depende de la inferencia de tipos del lenguaje

En la transparencia referencial:
*Los valores resultantes son inmutables
Los valores resultantes pueden cambiar
Los valores resultantes cambiarán dependiendo de la inferencia de tipos
Ninguna de las anteriores

Si quisiera simular un ciclo en Haskell:
*Debería utilizar recursividad.
Debería crear una función especificando la palabra clave repeat.
Debería usar guardas y condicionales.
No es posible simular un ciclo en Haskell

Cuando uso listas en Haskell:
*Puedo acceder a cualquier elemento de la lista usando el operador !!
Puedo recorrerla usando un ciclo
Únicamente puedo acceder a los elementos usando cabecera y cola
Puedo acceder a los elementos sólo con recursividad

Característica NO fundamental de los lenguajes funcionales:
Utilización de funciones polimórficas
Evaluación perezosa
*Construcción de abstracciones a través de funciones o relaciones
Utilización de tipos de datos genéricos

El paradigma funcional tiene diversas áreas de aplicación entre las cuales podemos enumerar:
Desarrollar aplicaciones en Tiempo Real, con muy poco tiempo de respuesta
*Resolver problemas que requieran demostraciones por inducciones.
Aplicaciones en cuya formulación existen reglas resueltas por un Motor de Inferencias.
Aplicaciones que usan exclusivamente el modelo MVC (Model View Controler)

Relativo a las funciones en Haskell:
En el Lenguaje Haskell no son de orden superior
Todos sus argumentos se evalúan previamente a la evaluación del cuerpo
*Podemos referenciarlas por su nombre, sin necesidad de conocer su estructura interna
Requieren ser definidas previo a su uso: Nombre, argumentos de entrada/salida

Relativo a las funciones de orden superior NO corresponde decir que:
Alguno de sus argumentos es una función o devuelve una función como resultado
*Alguno de sus argumentos es una función y siempre devuelven una función como resultado
Son útiles porque permiten capturar esquemas de cómputo generales (abstracción).
Los argumentos y resultados de estas funciones pueden ser a su vez funciones usadas como argumentos de otras funciones

Respecto a la evaluación perezosa:
Es una característica propia de todos los lenguajes funcionales
*El argumento de una función sólo se evalúa cuando es necesario para el cómputo
Al evaluarse un argumento se lo hace en forma completa, una única vez
Solo al manipular estructuras de datos 'infinitas' no podemos utilizarla

Respecto a las funciones en Haskell:
Las funciones juegan un papel no esencial
*Pueden ser argumentos o resultados de otras funciones o ser componentes de estructuras de datos
Son imprescindibles la definición de signatura y la implementación de su cuerpo
Ninguna de las anteriores

Haskell es un lenguaje de programación fuertemente tipado porque:
Exige la definición de los tipos de entrada /salida previa a la definición del cuerpo de la función
Permite que las funciones de orden superior tengan asociado un tipo de dato de retorno
*Posibilita asociar un único tipo a toda expresión bien formada

Permite que el usuario defina sus propios tipos de datos

Respecto a la inferencia de tipos de Haskell:
El programador debe declarar el tipo de la expresión, el sistema chequea coincidencia
*Si el programador declara el tipo de alguna expresión, el sistema chequea que el tipo declarado coincida con el tipo inferido
Haskell utiliza el tipo declarado por el programador, o lo infiere si el programador no lo declaró
La declaración de tipos en una expresión tiene propósito solo documentativo, Haskell lo infiere

Las expresiones let de Haskell son útiles para:
Restringir el efecto colateral de las variables globales a expresiones específicas
Permitir que a una variable se asocie a un tipo específico de datos
*Definir un conjunto de declaraciones locales
Permiten que determinadas funciones accedan a variables definidas en otras

Identifique qué característica NO corresponde a Haskell:
Utilización de funciones sobre elementos de primer orden
Utilización de funciones de orden superior
Utilización de funciones polimórficas
*Evaluación ansiosa.

Una función de orden superior es:
*Una función tal que alguno de sus argumentos es una función o que devuelve una función como resultado.
Una función tal que todos sus argumentos son siempre una función.
Una función que siempre devuelve una función como resultado.
Una función que debe devolver una función como resultado.

Los tipos básicos en Haskell son:
*Booleanos, Enteros, Flotantes y Caracteres
Booleanos, Flotantes y Caracteres
Booleanos, Cadenas y Caracteres
Cadenas, Flotantes y Enteros

Indique cuál de las siguientes características NO es una ventaja de la evaluación perezosa:
Manipulación de estructuras de datos infinitas
No requiere más (sino posiblemente menos) pasos que la evaluación impaciente
Manipulación de computaciones infinitas
*La evaluación de expresiones se efectúa antes de aplicar una función

Los tipos compuestos son aquellos cuyos valores se construyen utilizando otros tipos y son:
Funciones y tuplas
Funciones, entero y flotante
Funciones y caracteres
*Listas, funciones y tuplas

Cuando se trabajan con símbolos de operador es necesario tener en cuenta:
* La precedencia y la asociatividad
La precedencia o la asociatividad
Asociatividad y Simetría
Ninguna de las opciones anteriores es correcta

Una expresión con guarda posee la siguiente característica:
*Función que contiene guardas que requieren que se cumplan ciertas condiciones sobre los valores de sus argumentos
Función que contiene guardas que requieren que se cumplan solo una condición sobre los valores de sus argumentos
Función que contiene guardas que permiten trabajar con los valores de sus argumentos
Ninguna de las opciones anteriores es correcta

En el entorno WinHugs, selecciona lo que NO corresponda respecto al corte:
Prelude.hs es un fichero que se carga automáticamente al arrancar la ejecución deWinHugs, y contiene un conjunto de funciones que podemos utilizar
WinHugs permite desde su prompt evaluar expresiones sintácticamente correctas
El comando :load <filename> puede ser utilizado para cargar módulos desde un archivo especificado
* Los archivos que nosotros creemos con las definiciones e implementaciones de nuestras propias funciones, deben tener la extensión .pl

Selecciona lo que NO corresponda respecto a funciones:
Una función es una regla de asociación que relaciona dos o más conjuntos entre sí
Cuando tenemos la asociación de dos conjuntos, la función se define como una regla de asociación entre un conjunto llamado dominio con uno llamado codominio o imagen
Se dice que el dominio de una función son todos los valores que puede tomar el conjunto del dominio y que encuentra correspondencia en el conjunto llamado imagen
*Las funciones establecen relaciones entre dos conjuntos, permitiendo que un elemento del dominio pueda tener más de dos imágenes que le correspondan

Selecciona lo que NO corresponda respecto a la evaluación perezosa:
Haskell utiliza la evaluación perezosa
Uno de los beneficios es la posibilidad de manipular estructuras de datos “infinitas”
Es una evaluación en orden normal
* Consiste en evaluar todos los argumentos de una función antes de conocer si son necesarios

Selecciona lo que NO corresponda respecto a las funciones en Haskell:
*En la implementación del cuerpo de una función se debe escribir el símbolo ::
Las funciones en Haskell son objetos de primera clase
Una forma de escribir funciones en Haskell es mediante la definición de su signatura y la implementación del cuerpo de la misma
En la definición de la signatura se especifican los tipos de datos de entrada y salida

Selecciona lo que NO corresponda respecto a tipos de datos en Haskell:
Haskell es un lenguaje de programación fuertemente tipado
Cada tipo de dato tiene asociadas un conjunto de operaciones que no tienen significado para otros tipos de datos
*El sistema de tipos en Haskell no permite detectar errores en expresiones ni en definiciones de función
Cualquier expresión a la que no se le pueda asociar un tipo es rechazada como incorrecta antes de la evaluación

Selecciona lo que NO corresponda respecto al sistema de inferencia de tipos en Haskell:
El compilador contiene un algoritmo que infiere el tipo de las expresiones
Si el programador declara el tipo de alguna expresión, el sistema chequea

que el tipo declarado coincide con el tipo inferido
Los sistemas de inferencia de tipos permiten una mayor seguridad, evitando errores de tipo en tiempo de ejecución y una mayor eficiencia, evitando realizar comprobaciones de tipos en tiempo de ejecución
*El programador está obligado a declarar el tipo de las expresiones

En el estándar Prelude.hs de Haskell, el operador (-), es:
*Asociativo a la izquierda
Asociativo a la derecha
No asociativo
Asociativo a la derecha y a la izquierda