RPA

▼ ¿Cuál es la definición de computadora?

una computadora es un sistema digital con tecnología microelectrónica compuesta por estos tres elementos:
1- CPU (del inglés Central Processing Unit) o Unidad Central de Procesamiento.
2- Memoria.
3- Dispositivos de Entrada y Salida.
Todos estos elementos interconectados por un canal de comunicación (llamado “bus”).

▼ ¿Qué diferencia hay entre un lenguaje de programación y un programa de computadoras?

Un lenguaje de programación (en inglés programming language) es un lenguaje artificial creado para expresar procesos que pueden ser llevados a cabo por computadoras. En cambio un programa de computadoras (en inglés computer program), es un conjunto organizado de instrucciones que están escritas en un lenguaje de programación, y al ser ejecutadas por una computadora resuelven una tarea específica.

▼ ¿Qué diferencia hay entre código fuente y código ejecutable?

El texto de un programa de computadoras se conoce como código fuente. El código ejecutable es una secuencia de código que la computadora ejecuta directamente al ser invocado (sin necesidad que el compilador esté presente). Generalmente de extensión EXE o COM.

▼ ¿Qué hace un compilador?

Un compilador (en inglés compiler) es un programa que traduce el código fuente de un programa, que está escrito en un lenguaje de programación, a otro lenguaje de programación (típicamente lenguaje de máquina) permitiendo generar código ejecutable (executable code).

Hardware: El hardware es el conjunto de los componentes que integran la parte material y tangible de una computadora: chips, circuitos, cables, periféricos de todo tipo y cualquier otro elemento físico involucrado.

Software: El software es el conjunto de todos los programas de cómputo, documentación y también los datos asociados que forman parte de un sistema de computación.

Un bit es la unidad mínima para representar información. Permite almacenar únicamente dos valores (generalmente representados con 0 y 1).

Byte: es una unidad de medida utilizada en informática. Un byte equivale a 8 bits y permite representar 256 valores.

Problema: “planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos”.

Un problema se distingue de un ejercicio. En un ejercicio, se busca encontrar una solución a una consigna aplicando una fórmula, o un método conocido. En un problema, en cambio, no resulta evidente el camino a seguir, ya que no se dispone a priori de una fórmula o método para aplicar. Además, puede haber varios caminos alternativos que permitan resolverlo. La resolución de un problema requiere aplicar y vincular conocimientos previos, probablemente de áreas diferentes, buscando nuevas relaciones.

Algoritmo: es la especificación de una secuencia de pasos u operaciones, que al ser ejecutadas permiten resolver un problema. Un algoritmo debe tener un único punto de inicio y al menos un punto final; y todos sus pasos deben estar expresados con operaciones comprensibles para quién las ejecutará (a las cuales llamamos primitivas).

Una primitiva es: una operación conocida, utilizada en un algoritmo y considerada como básica.

Una traza es :: una simulación de la ejecución real de los pasos de un algoritmo, en la cual se lleva cuenta de los movimientos realizados y los cambios que se producen en los elementos o datos involucrados.

Los casos de prueba son: situaciones concretas en las cuales se puede aplicar el algoritmo desarrollado, y se utilizan para verificar que el algoritmo se comporta de la manera esperada. Según sea el problema para el cual se usará el algoritmo, los casos de prueba pueden incluir valores articulares para los datos u otros elementos que están involucrados en la solución del problema. Cada caso de prueba debe incluir cuál es el resultado esperado. En resumen, un caso de prueba describe una situación particular en la que se puede aplicar el algoritmo y además, cuál es el resultado esperado.

Diferencias entre constantes y variables en Pascal: Al declarar una constante se le asigna un valor que no puede cambiar durante la ejecución del programa. Ese valor es fijo para todo el programa y el tipo de dato asociado depende del valor elegido. Cuando

declaras una variable, lo haces justamente para que su valor varíe y pueda cambiar (sino, usarías una constante). Además, puede ser que el valor lo determine alguien dentro de 2 o 3 años. Por lo tanto, al momento de escribir el código fuente, no es posible predecir qué valor exacto tendrá.

Tipo de Dato: Define el conjunto de valores posibles que puede tomar un elemento de un programa (ya sea una variable, constante, etc.), también define las operaciones que pueden aplicarse sobre esos valores, y, además, cuál es la representación interna para su almacenamiento en memoria.

De ejemplos de tipos de datos predefinidos en Pascal: Por ejemplo, el tipo de dato (predefinido) REAL en Pascal define que el conjunto de valores que puede tomar una variable real es un subconjunto de los números reales, que se puede usar las operaciones de adición, sustracción, multiplicación, división (entre otras) y que el espacio que usa para cada real es 6 bytes.

Error de compilación: (también llamado error en tiempo de compilación) es un error detectado por el compilador al momento que se está realizando la compilación de un código fuente. Son los más fáciles de encontrar y resolver.

Error de ejecución: (también llamado error en tiempo de ejecución) ocurre cuando durante la ejecución del programa hay una situación anormal que provoca que la ejecución se corte abruptamente.

Error de programación: (también llamado “bug” o error lógico), es un error en la lógica del algoritmo o programa el cual causa que no se resuelva correctamente la tarea que debe hacer el programa.

Diferencia entre read y readln: read(algo) Lee un elemento del buffer y se lo asigna a la variable algo. Si no hay ningún elemento en el buffer espera a que se carguen valores y se presione enter. En cambio readln busca un enter en el buffer, y como efecto colateral, vacía el buffer. Si no encuentra ningún elemento en el buffer espera a que se carguen valores y se presione enter.

▼ Los identificadores reservados de Pascal tienen dos características muy importantes

- Ya tienen un significado asignado en el lenguaje.
- Ese significado es fijo y no se puede cambiar.

▼ Los identificadores predefinidos en Pascal que tienen estas características

- Tienen un significado pre-establecido.
- Si fuera necesario, el significado puede cambiarse.

ej. REAL tipo de dato predefinido para identificar elementos que
representen números reales.
write y writeln primitivas predefinidas, permiten escribir texto o
datos en la pantalla del dispositivo.

▼ Los identificadores definidos por la persona que escribe el código fuente y tienen estas características

- El significado lo establece la persona que escribe el programa.
- El significado puede cambiarse.

Cuando declarás una variable, todavía no le diste ningún valor inicial. Simplemente declarás su existencia, indicás que tipo será y, por lo tanto, cuanta memoria usará. Pero declararla no le da un valor inicial. En realidad, tiene un valor desconocido y aleatorio que podría ser cualquier valor.

Los diagramas sintácticos (en inglés Syntax diagrams or railroad diagrams) son: una forma gráfica de especificar la sintaxis de un lenguaje de programación. Se los llama también “railroad diagrams” (diagrama del ferrocarril) porque cuando los ves se asemejan a un mapa de las vías que conectan estaciones. Y es de esa manera que hay que usarlos, recorriendo el diagrama siguiendo las flechas que muestran los caminos posibles.

▼ ¿Qué es una *sentencia compuesta* en Pascal?

Una “proposición” que se llama sentencia compuesta y que permite que una secuencia de sentencias sea considerada como una sola sentencia. Para poder tener una sentencia compuesta ponés begin, todas las sentencias que quieras separadas por punto y coma; y luego un end.

▼ Pautas de buena programación.

Pauta #1: comentar adecuadamente el código fuente. Escribiendo comentarios en lugares adecuados para que otra persona que lea tu código (que podés ser vos en el futuro) entienda las decisiones que tomaste al resolver el problema. Podés incluir como comentario tu algoritmo y los casos de prueba que usaste.
Pauta #2: elegir nombres significativos para los identificadores. Ya sean variables, constantes, o cualquier otro elemento del código fuente en donde puedas elegir vos el nombre.
Pauta #3: usar el tipo de dato más adecuado. Usar un tipo de dato adecuado permite claridad y abstracción: dos conceptos fundamentales en el desarrollo, mantenimiento y futuras actualizaciones del software. Además de ayudar a la lectura del código fuente por humanos, el compilador puede generar un mejor código y mejores controles.

Pauta #4: indentar el código. Al programar con condicionales y con repeticiones, y muchas veces con anidamiento de una condición con otra condición o una repetición (como te mostré sobre el final del capítulo de algoritmos), una buena organización del código fuente usando indentación es fundamental para entender el código.

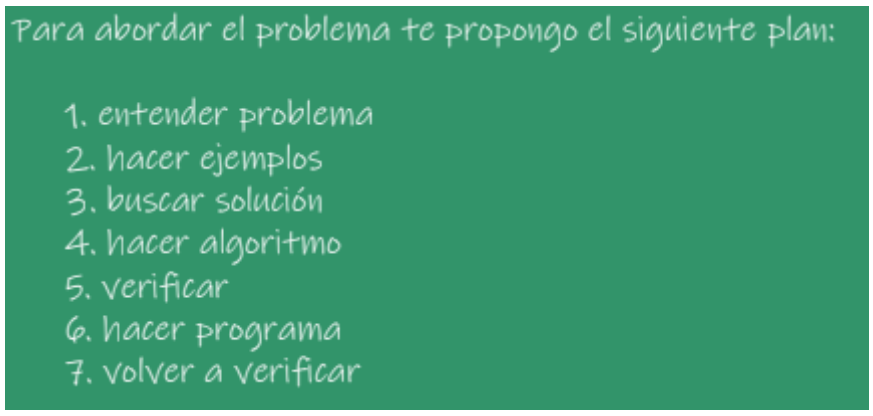
Buffer: En computación, un buffer (amortiguador) es un espacio de memoria en el cual se almacenan datos de manera temporal. En general, se usa para amortiguar la diferencia de velocidad de dos o más dispositivos entre los cuales se están transfiriendo datos.

La sentencia while es más general que la sentencia for, ya que todo lo que se puede implementar con un for, lo podés implementar con un while, programando el incremento (o decremento) de la “variable de control”.

▼ ¿Por qué while y repeat son inversos?

La razón por lo cual las condiciones de un while y un repeat son “inversas” está en la semántica de cada sentencia: un while se repite si la condición es verdadera, y en un repeat se repite si la condición es falsa. Además un while repite “0 o más veces”, y un repeat “1 o más veces”.

Debugging (depuración): se refiere al proceso metodológico de buscar y reducir el número de errores o defectos (bugs) de un programa, con el objetivo de lograr que el programa tenga el comportamiento esperado.



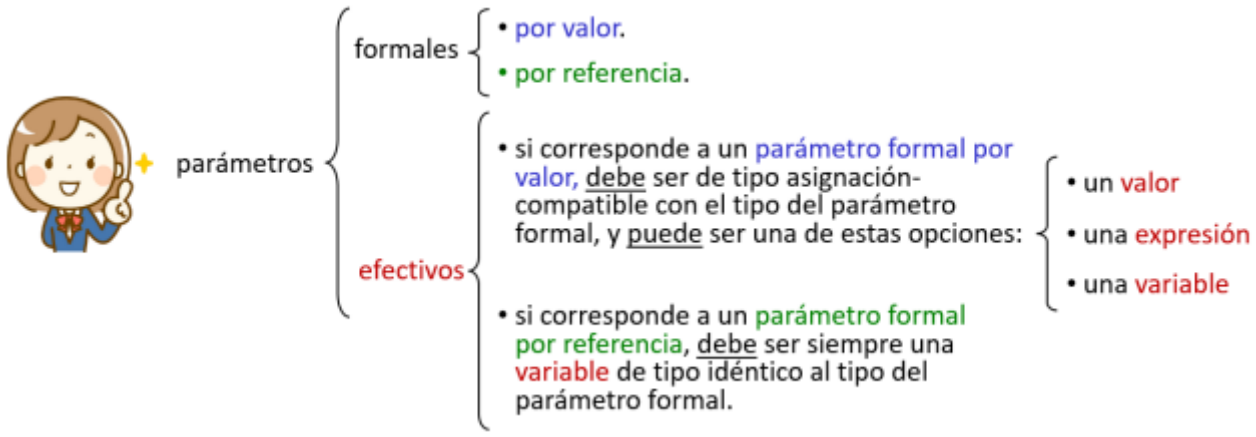
La mayor similitud entre funciones y procedimientos, es que ambos permiten la implementación de nuevas primitivas. La mayor diferencia entre funciones y procedimientos es como se realiza la llamada a cada uno. Una función se debe llamar en una expresión, y cuando se retorna de la llamada a la función se sigue ejecutando la misma sentencia en donde está la llamada. En cambio, un procedimiento se llama como una sentencia (como write o read) y al retornar de la ejecución del procedimiento se ejecuta la sentencia siguiente a la llamada.

Las funciones tienen que: tener un propósito claro y deben resolver un problema puntual. Siempre deben retornar un resultado y ese resultado será usado en una expresión. Las funciones siempre deben retornar un valor para cualquier valor de entrada, de lo contrario es un error de programación.

En cualquier función o procedimiento está permitido: usar constantes, procedimientos, o funciones que fueron declarados en el programa que los contiene. También se pueden usar variables propias o parámetros. Sin embargo, se considera un error de programación, que un procedimiento o función utilice o modifique una variable que no fue declarada como propia.

En Pascal hay dos clases de parámetros:

- Parámetros por valor: reciben una copia del valor del parámetro efectivo y, por lo tanto, se los utiliza solamente para entrada de datos. Si los parámetros son por valor, los parámetros efectivos pueden tener variables, expresiones o valores (siempre que sean del tipo adecuado).
- Parámetros por referencia: para distinguirlo se antepone el identificador var al parámetro. En esta clase de parámetros se crea una referencia entre el parámetro formal y el parámetro efectivo que corresponde, y como es una referencia, todo cambio que se realice en el parámetro formal afectará al parámetro efectivo. En este caso los parámetros efectivos solamente pueden tener variables del mismo tipo que el formal.



El entorno de referencia de un bloque B está formado por los siguientes cuatro entornos:

1. El entorno local: conjunto de identificadores (parámetros formales, constantes, tipos, variables, el nombre de los procedimientos y funciones) declarados dentro del bloque B.
2. El entorno global: conjunto de identificadores declarados en el bloque del programa principal.
3. El entorno no-local: conjunto de identificadores declarados en los bloques que contienen al bloque B, exceptuando al global.
4. El entorno predefinido: conjunto de identificadores predefinidos por el compilador y disponible para todo programa (Ejemplos: maxint, char, integer, read, write, readln, writeln, eof, eoln).

Cuando se hace referencia a un identificador en un bloque B:

1. primero, se busca en el entorno de referencia local del bloque B,
2. luego, se busca en el entorno de referencia no local de B,
3. luego, se busca en el entorno de referencia global, y
4. finalmente, se busca en el entorno de referencia predefinido

Si hay identificadores con el mismo nombre en diferentes entornos, entonces uno oculta al otro de la siguiente manera:

1. Un identificador de nombre N en un entorno local oculta a todo identificador del mismo nombre N en otro entorno (no-local, global, predefinido)
2. Un identificador de nombre N en un entorno no-local, oculta a otro N global o predefinido,
3. Un identificador global de nombre N oculta a uno predefinido de nombre N.

Un identificador es referenciable en un bloque, si es parte de su entorno de referencia y no está oculto.

Un identificador es visible, si es referenciable.

El alcance de un identificador D, son aquellas sentencias (o bloques) del programa donde el identificador D es visible.

Los identificadores que están declarados en el entorno global, son visibles en todos los bloques del programa (salvo que otro identificador en el mismo nombre lo oculte. Como el entorno predefinido es el mismo para todos los programas, entonces los identificadores del entorno predefinido (integer, real, read, write, etc.) serán visibles en cualquier bloque de cualquier programa (salvo que sean ocultados por otro identificador del mismo nombre).

Recursión es la forma en la cual se especifica un proceso basado en su propia definición. Un algoritmo es recursivo si se define en términos de sí mismo.

Un planteo recursivo es una solución a un problema donde: (a) se indica un caso base (también llamado caso trivial) que no se define en términos de sí mismo, y, además, (b) se indica un caso general (también llamado caso recursivo) donde se hace referencia a sí mismo, pero debe ser sobre una instancia reducida del problema.