UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

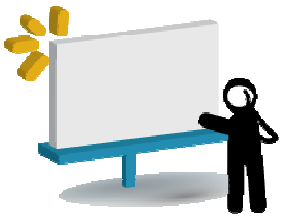
Centro de e-Learning

FUNDAMENTOS DE LA PROGRAMACIÓN

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

MÓDULO 1 - UNIDAD 3

Algoritmos



Presentación:

En esta Unidad continuamos desarrollando lo que llamamos la "forma de pensar" del programador. Para eso veremos algunos conceptos básicos que nos permitirán ir delineando el trabajo cognitivo involucrado en las tareas de programación.

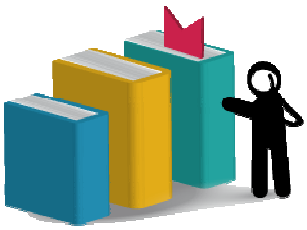
Adicionalmente, veremos un posible método con un proceso asociado para el planteo de una posible solución a un problema dado.

También analizaremos las distintas formas de representar algoritmos.

Objetivos:

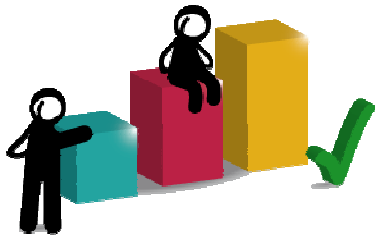
Que los participantes:

- Comprendan e incorporen los principales aspectos necesarios relacionados a la "forma de pensar" del programador
- Comprendan e incorporen las prácticas de razonamiento y resolución de problema necesarias para desarrollar algoritmos
- Comprendan los tipos de algoritmos y la forma de representarlos
- Analicen un caso integrador de los temas vistos



Bloques temáticos:

1. Aplicando la lógica
2. Resolución de problemas
3. Algoritmos
    - 3.1 Definición
    - 3.2 Representaciones
    - 3.3 Descripciones de los algoritmos
4. Caso integrador
5. ANEXO - Otros algoritmos resueltos



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

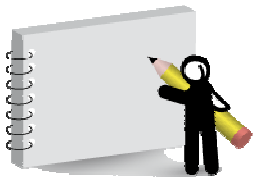
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

* El MEC es el modelo de E-learning colaborativo de nuestro Centro.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Aplicando la lógica

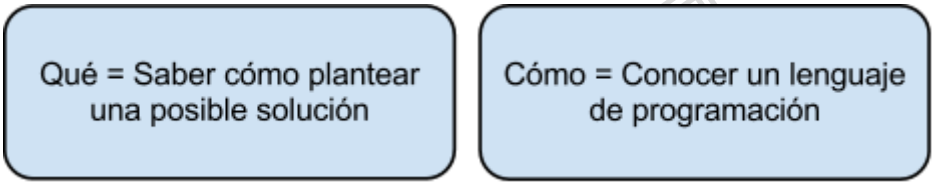
Si bien no es objetivo del curso seguir profundizando aspectos propios de la lógica en sí, es importante contar con una base teórica y práctica que fundamente una de las principales armas del programador: el pensamiento lógico orientado a la resolución de problemas. Como complemento de los aspectos formales analizados en la Unidad anterior, que explican los basamentos de las habilidades y conocimientos necesarios para la programación, en esta Unidad nos permitiremos plantear una serie de métodos propios, más informales, para llevar estas prácticas, justamente, a la práctica.

Un aspecto fundamental para todos aquellos que se inician en la programación es entender el "**qué**" y el "**cómo**" de la resolución de un problema planteado, que es lo que informalmente nos referimos a la "forma de pensar del programador".

En este contexto, el "**qué**" implica plantear una solución a un problema determinado, a través del planteo de acciones que lleven a una posible solución final. Esto se puede hacer tanto sobre una hoja o pizarra, o con un software especializado para realizar diagramas que expliquen cómo resolver un problema, aunque en forma gráfica, visual.

El "**cómo**" toma como entrada el "**qué**" para llevar ese conjunto de acciones o pasos expresados en una hoja, pizarra o gráfico a una serie de instrucciones en el lenguaje de programación que se haya elegido.

Aquí es donde podemos plantear los siguientes paralelismos:



Está claro que el "**qué**" es lo primero que deberemos atender, ya que de poco vale conocer a la una herramienta (cualquiera) si no se la sabe usar correctamente.

Es aquí donde surge la reflexión y una crítica común por parte de los profesionales de la programación a los exámenes de certificación en lenguajes de programación: si bien no es posible ser absoluto en cuanto a la postura, es evidente que muchas compañías u organizaciones utilizan estas certificaciones con fines netamente comerciales, perdiendo el foco de lo que realmente deberían ser las certificaciones iniciales en programación, y se orientan a comprobar el "conocimiento del lenguaje X" por parte del candidato a certificar por sobre "el conocimiento en programación con el lenguaje X".

Retomando la importancia del "**qué**", con el fin de sentar las bases de "cómo se programa" por sobre "cómo se maneja cierto lenguaje", podemos afirma que el correcto razonamiento y planteo de la solución es un proceso fundamental que debe realizarse antes de comenzar a resolver en forma programática un problema. De esta forma, el programador va a evitar una clara pérdida de tiempo y necesidad de retrabajo al tener una solución robusta y probada, al menos en forma teórica, claro está.

Aquí es donde entra en juego la lógica en la programación.

Ante todo, podemos decir que la lógica es la ciencia que estudia el proceso de razonar y los procesos asociados a la actividad del razonamiento. Una definición simple del razonamiento podría ser el hecho de tener que organizar las ideas (recordar el concepto de proposiciones o enunciados visto antes) para llegar a una conclusión sobre un problema determinado.

Este proceso de organización no es ni más ni menos que una sucesión de pasos interconectados que, en su conjunto, conforman una proposición o planteo de una posible solución.

Una lección importante en la lógica de todo proceso de resolución de problemas consiste en subdividir un problema en “problemas más pequeños” en forma sucesiva, de manera de poder llevar el problema a un esquema de "tareas atómicas". **Esto significa que cada “subproblema” pueda ser resuelto con una única actividad o tarea que sea lo más simple posible.** De esta forma, organizando y priorizando estas tareas, se podrá atacar cada uno en forma individual y luego, en su conjunto y una vez completados todos, se llegará a la solución global al problema inicialmente planteado.

Este esquema de división o desglose de trabajo es sumamente útil en el trabajo diario del programador y hasta se vuelve automático e inconsciente, llegando a ser parte de la “forma de pensar del programador”.

Tomando esta estrategia como premisa, nos adentramos en el parte final de la lógica, referida al “Proceso”, donde las ideas son ordenadas y organizadas, y al que nos referiremos como resolución de problemas, siendo éste el “**qué**”.

2. Resolución de problemas

La resolución de problemas es una actividad íntimamente relacionada con el proceso de razonamiento, el cual es aplicado constantemente (y en cierto punto, inconscientemente, como se dijo antes) por los programadores. Esto no quita que se trate de una actividad que deba ser comprendida en su esencia, que deba ser ejercitada para su perfeccionamiento y posteriormente puesta en práctica, y finalmente incluida en el día a día de los desarrolladores.

Tomemos como ejemplo práctico un problema, como ser la necesidad de modelar conceptualmente la famosa serie de Fibonacci (no vamos a ser muy puristas desde lo matemático, así que nos permitiremos tomarnos algunas licencias para facilitar la explicación).

a) Inicialmente, sabemos que la serie de Fibonacci:

a.1) Es una serie de números

a.2) Son números naturales

a.3) Es infinita

a.4) Un ejemplo finito podría ser: 1, 1, 2, 3, 5, 8, 13, 21 y 34

b) Conociendo cómo "funciona" la serie, podemos decir que:

b.1) Comienza en 1

b.2) Se incrementa exponencialmente

b.3) La serie aumenta al agregarse de a un nuevo valor en la misma

b.4) El nuevo valor surge de la suma del último y penúltimo valor de la serie

c) Finalmente, una posible interpretación podría ser:

c.1) Siendo “x” el último valor e “y” el anteúltimo, “z”, el nuevo valor de la serie resultará de **"z = x + y"**

c.2) Repetir esto último en forma sucesiva 8 veces (para cumplir con el ejemplo finito antes mencionado)

En este proceso, podemos identificar una serie de etapas y actividades llevadas a cabo en las mismas. Podemos pensar que este proceso consiste en las siguientes etapas:

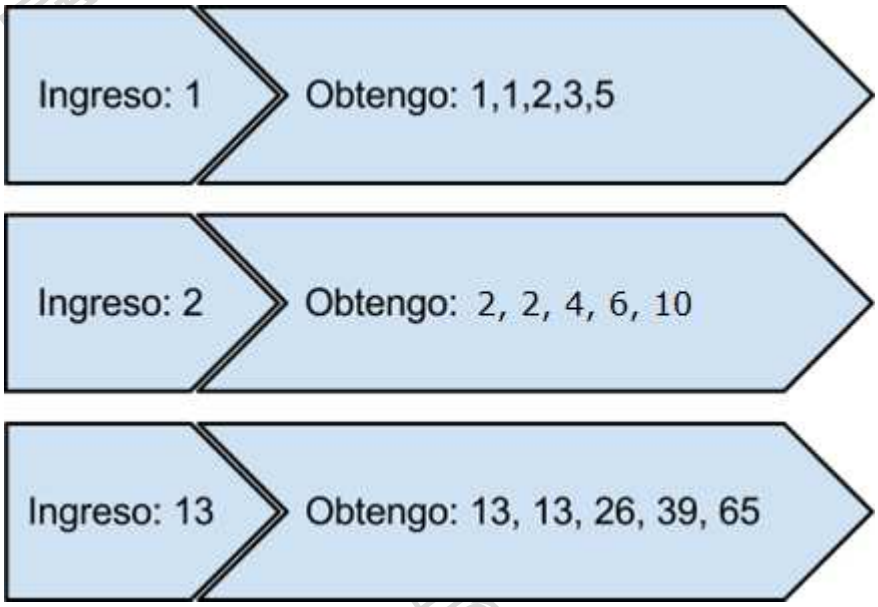
- **ENTRADA**:
  - Tal como hiciéramos en el punto a), comenzamos por entender el problema. En esta instancia, es importante enfocarse en el PROBLEMA por sobre la SOLUCIÓN.
  - Para esto debemos recopilar información y detalles de la situación que estamos analizando, identificando los elementos que componen el problema y posteriormente analizándolos. Aquí es donde volvemos a recurrir al recurso del “desglose” del problema: tomar el todo, subdividirlo en tareas individuales pequeñas.

- Posteriormente pasaremos a priorizar y ponderar estos elementos, dándoles una importancia (en cuanto a la información que aportan) y una urgencia (la cantidad de atención que debemos prestarles). En nuestro caso, el elemento de análisis que mayor información nos da es el ejemplo de números que componen la primera parte de la serie, según lo que dice el punto a-1).

- **PROCESO**
  - En el punto b), nos enfocamos en generar opciones de solución. En este punto dejamos de enfocarnos en el PROBLEMA para buscar la mejor alternativas de SOLUCIÓN. Enumeramos las opciones que nos permiten llegar a una solución en base a lo planteado en la fase anterior.
  - Posteriormente, repetimos la actividad de priorizar las opciones para enfocarnos en las que nos acercan más a la construcción de la solución.
  - Posiblemente los enunciados a los que lleguemos en este punto estén relacionados entre sí. Ellos deberán ser agrupados según su solución, de manera que un grupo de enunciados esté orientado a una única alternativa de solución. Esto suele ocurrir en caso de encontrar más de una solución posible (lo cual sucede normalmente en los problemas complejos o en el caso que en la fase de Entrada no se tenga un análisis suficientemente cerrado, por ejemplo, por falta de información).
  - Al final, el Proceso termina cuando se selecciona la mejor opción de solución y se cuenta con un conjunto de conceptos orientados a comprender el problema para llegar a una solución.

- **SALIDA**
  - Finalmente, tal cual lo mencionado en el punto c), en la fase de Salida se plantea la mejor solución para un problema, el cual fue “desmenuzado” y analizado en la fase de Entrada y comprendido en su esencia en la fase de Proceso.
  - **Aquí es donde se aplica y concreta la solución**. En nuestro caso de la serie de Fibonacci, lo que terminamos obteniendo es una fórmula matemática que representa en forma genérica la serie, con un grupo de aclaraciones que le dan un contexto en particular.
  - El proceso se completa con una actividad que no siempre se tiene en cuenta y que resulta de gran importancia: la evaluación de los resultados.
  - En nuestro caso (favorablemente sencillo) la evaluación consistiría en comprobar que la fórmula presentada cumple con la serie, si se la ejecuta en 8 oportunidades en forma sucesiva. En este caso es sencillo de comprobar la corrección de la solución planteada. En caso de problemas complejos, con múltiples variables, contextos y actores, la verificación del planteo no es siempre tan sencilla. Lo que hacemos en este caso es realizar lo que se llama **casos de pruebas**, modificando los datos de entrada y comprobando que el resultado obtenido es el esperado.
  - Tomemos por ejemplo un cambio en el requerimiento inicial: “representar los primeros 4 números de la serie de Fibonacci a partir de un número dado”.
  - Con este cambio de requerimientos deberíamos estar seguros de que la solución planteada va a funcionar si el número dado es, por ejemplo, un 1, un 2 o un 13.

Nuestros casos de pruebas deberían ser:



[Siendo estrictamente puristas, diríamos que esta última serie de números no pertenece a Fibonacci, pero dejamos esta licencia para graficar el ejemplo].

El proceso finalizaría dando por cerrado el problema, en caso de cumplir con los casos de prueba, o volviendo al inicio, en caso de haber detectado una falla en el planteo

ACTIVIDAD 1:

Utilizando el proceso antes mencionado, buscar una solución similar a la Fibonacci que cumpla con lo expresado en las siguiente series:

- Serie 1: 1, 2, 3, 4, 5,6,7,8,9,10

- Serie 2: 10, 8, 6, 4

- Serie 3: 100, 100, 300, 700, 1.700, 4.100, 9.900

Aclarando:

- Cuál es la expresión matemática genérica que cumple con la serie

- Cuáles son las aclaraciones contextuales (las llamaremos precondiciones) necesarias para que la expresión cumpla con la serie

Comparta sus respuestas en el foro de la Unidad.

3. Algoritmos

3.1 Definición

Prácticamente encontraremos tantas definiciones de algoritmo como autores que escriban sobre el tema. Una forma posible de describir un algoritmo sería decir que se trata de **una abstracción conformada por una serie de instrucciones que apuntan a resolver un problema determinado mediante el planteo de una serie de pasos, en los cuales, a través de una serie de datos de entrada, se obtiene como salida una solución total o parcial**. Esto último puede ocurrir en caso de que se combinen varios planteos para llegar a una determinada solución, a pesar de lo cual cada uno de dichos planteos no deja de ser un algoritmo en sí mismo.

En general, se pueden tomar ciertas características o propiedades en común con respecto a los algoritmos:

- **Secuencialidad**: un algoritmo avanza o se ejecuta paso a paso, por lo que se define como una secuencia con pasos concretos intermedios, en los cuales se puede individualizar y analizar cada uno de estos paso, por su estado, entradas y salida (en caso de tenerla).
- **Atomicidad**: cada uno de los pasos concretos intermedios se representan a través de una acción, la cual resuelve uno y sólo un problema simple a la vez.
- **Abstracción**: como dijimos, por definición, un algoritmo es una representación abstracta, por lo que debe ser independiente de su implementación. Para ejemplificar este punto, nos podemos remitir nuevamente a nuestra solución de Fibonacci:
    - Una posible solución es crear un programa que directamente muestre los números 1,1,2,3,5..., en donde no tendríamos un algoritmo, ya que no es una solución abstracta sino concreta.

- La solución válida sería contar con una fórmula que resuelva en forma genérica la serie:
  "anteúltimo_número + último_número = nuevo_número_de_la_serie"

A modo de resumen integrador, podemos decir que un algoritmo es cualquier conjunto de instrucciones ordenadas con el fin de que se ejecuten paso a paso, donde cada paso tiene un estado que puede ser descripto unívocamente, que sea de carácter genérico y que realice un conjunto finito y acotado de modificaciones a un estado en particular.

3.2 Representaciones

Existen numerosas formas de representar un algoritmo, de las cuales algunas serán presentadas en este curso.

- **Lenguaje natural**: son expresiones realizadas en forma narrativa, que tienen como ventaja una gran capacidad de abstracción y cobertura (dadas por el mismo lenguaje), aunque tienen como desventaja el riesgo de resultar ambiguas, poco precisas y extensas.
- **Seudocódigo**: podemos decir que es un subconjunto del lenguaje natural, en el cual por consenso se limitan las palabras y se forman estructuras estándares, por lo cual tienden a evitar los problemas antes mencionados.
- **Diagramas de flujo**: en esencia, tiene las mismas características que el seudocódigo, aunque se trata de representaciones gráficas.
- **Lenguajes de programación**: es un conjunto aún más reducido de palabras y estructuras que tiene como principal diferencia con el seudocódigo y los diagramas de flujo que es genérico, es decir, que es independiente de una implementación en particular. La principal ventaja es que puede obtener un “producto” o programa que realmente “haga algo” por tener la posibilidad de ser ejecutado.

Los tres primeros tipos de representación serán los analizados en este curso, dejando la implementación en un lenguaje en particular fuera del alcance del curso. Esto se debe a la necesidad mencionada al principio del curso de crear el contexto para "la forma de pensar" que debe desarrollar, practicar y tener cada programador.

3.3 Descripciones de los algoritmos

Existen diversas formas de describir los algoritmos. Si los vemos desde el punto de vista del nivel de detalle, podemos identificar los siguientes 3 niveles, comenzando desde un detalle menor, en aumento:

- **Descripción de alto nivel**: se investiga el problema y se realiza una descripción narrativa, comúnmente acompañada de gráficos sin entrar en detalle. Normalmente de trata de definiciones ad-hoc, que no siguen un estándar particular. Su objetivo es presentar una solución para un tipo de audiencia o destinatarios inexpertos. Este es el tipo de notación que utilizamos para describir la serie de Fibonacci y en la que se presentan los ejemplos de esta unidad posteriormente.

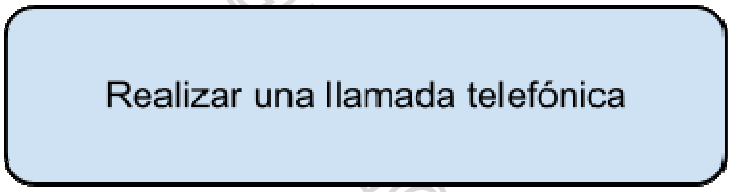
- **Descripción formal**: el algoritmo es descripto en una secuencia de pasos, los cuales son representados a través de seudocódigo y diagramas de flujo. De esta forma se obtiene una descripción de la solución con un nivel de detalle suficiente para que sea comprendida reduciendo ambigüedades y brindando más información. Este será el nivel de detalle utilizado en el presente curso.

- **Implementación**: se trata del algoritmo llevado a un lenguaje de programación en particular. Al tratarse del algoritmo concreto, es el máximo nivel de detalle al que se puede llegar.

4. Caso integrador

A continuación, desarrollaremos un caso integrador que abarca los puntos vistos en esta unidad. Es importante aclarar que las actividades del programador no siempre se centran en resolver diversos problemas matemáticos, como lo fue el caso de la serie de Fibonacci. De hecho, para ser claros, esto ocurre la menor parte de las veces. Lo que sí es común encontrar son problemas o situaciones (si se quiere) más mundanas. Como dijimos al principio de este curso, la programación es, por definición, la realización de un conjunto de actividades que tienden a modelar (crear un modelo) en forma abstracta de una realidad en un determinado contexto.

Pasemos entonces a tomar una situación que permita ser modelada y que resulte más familiar. Supongamos que se nos pide realizar un algoritmo que cumpla con el siguiente enunciado:



1. No pensamos todavía en el teléfono, ni en los números ni en la llamada en sí. Comenzaremos por obtener más información sobre lo que debemos hacer. Para ello, encaremos la resolución de este problema desde un punto de vista lógico, tal como dijimos, como un proceso en el que el primer paso es obtener información de contexto, y plantear y ordenar ideas.

a. **Información de contexto**: restringiremos el problema a modelar de forma tal que no quede duda sobre el alcance de las actividades que debemos desarrollar. En nuestro caso, consultando obtenemos la siguiente información adicional:
   i. Debemos modelar el proceso en el cual una persona realiza una llamada telefónica.
   ii. La persona no es nadie en particular, el algoritmo debe permitir que, cualquiera sea la persona designada, se cumpla con el proceso. Dicho de otra manera, el algoritmo debe funcionar para cualquier persona.
   iii. Mismo caso para el teléfono, siempre y cuando sea un celular (teléfono móvil). No nos preocupa qué teléfono sea, siempre y cuando tenga como mínimo:
      1. Botón “Llamar”
      2. Botones del 0 al 9
      3. Botón “Finalizar llamada”
   iv. El algoritmo se da por finalizado cuando se termina la llamada. El funcionamiento del teléfono, la llamada en sí, las líneas telefónicas, centrales, etc., quedan excluidos del problema a modelar.

2. Ya tenemos nuestras ideas principales, las validamos y las ordenamos. Le dimos un contexto general al problema. Nos encargamos de que no haya “cabos sueltos”, es decir, circunstancias que podamos haber olvidado o dejado de lado. **Hasta ahora no pensamos en la solución, ni en cómo llegar a ella.**

3. Estamos listos para enfocarnos en el “QUÉ”. Esto es: plantear una solución para el problema dado, ya contextualizado. Lo haremos siguiendo el proceso definido de entrada-proceso-salida.
    a. **Entrada**: nos abocamos a entender el problema, inicialmente comprobando que conocemos y entendemos los elementos que lo componen. Para esto, desglosaremos los elementos componentes y las actividades que se realizan: una persona, un teléfono, la llamada (que puede ser subdividida en marcar, llamar y colgar). Vemos que son todos aspectos que nos resultan comprensibles y familiares, por lo que podemos avanzar con la solución del problema en sí.
    b. **Proceso**: comenzamos a buscar alternativas de solución. Es importante remarcar que posiblemente esta parte del proceso general deba repetirse sucesivamente hasta llegar a la solución que resulte óptima. A continuación enumeramos las actividades con las que deberá cumplir el algoritmo “Llamar por teléfono”:
        i. Tomar el teléfono
        ii. Marcar un número
        iii. Realizar la llamada
        iv. Mantenerunaconversación
        v. Colgar
        vi. Fin del proceso

Nuevamente, comprobamos que no quedan dudas sobre los pasos básicos para resolver el problema. En este punto surge el hecho de “Mantener una conversación” como punto intermedio entre realizar la llamada y colgar. Para esto, indagamos más en el contexto y llegamos a la conclusión (previa consulta con quien nos plantea el problema) de que debemos tomar en cuenta que existirá una conversación con un intercambio de 2 mensajes como máximo entre emisor y receptor de la llamada, cada uno.

También puede surgir la duda a qué clase de teléfono se realizará la llamada: ¿Un interno de pocos dígitos? ¿Otro número móvil? ¿A un fijo? Nuevamente, consultando llegamos a la conclusión de que solamente se deben poder realizar llamadas locales a un número fijo de 8 dígitos.

Damos por comprendido totalmente el problema ya con una aproximación a la solución. Pasemos ahora a plantear la solución completa y definitiva.

c. Salida: como salida del proceso obtendremos una representación utilizando lenguaje natural para describir un algoritmo que cumpla con la secuencialidad y la abstracción, y cuyos pasos intermedios tengan un número limitado de estados intermedios. Llamemos a este algoritmo “Llamar por teléfono – versión 2”:

EL ALGORITMO:

i. Dada una persona, a quien llamaremos emisor
ii. ¿El emisor tiene un teléfono en la mano?
    1. En caso de no tenerlo, lo toma.
    2. En caso de tenerlo, continúa el algoritmo.
iii. El emisor ingresa el número:
    1. Ingresa primer dígito
    2. Ingresa segundo dígito
    3. Ingresa tercer dígito
    4. Ingresa cuarto dígito
    5. Ingresa quinto dígito
    6. Ingresa sexto dígito
    7. Ingresa séptimo dígito
    8. Ingresa octavo y último dígito
iv. Se realiza la llamada apretando el botón “Llamar”
v. Se realiza la comunicación. Esto ocurre cuando la persona que recibe la llamada, a quien identificaremos como receptor, emite un primer mensaje:
    1. Se concreta la comunicación entre emisor y receptor
    2. El receptor atiende, emitiendo el primer mensaje
    3. El emisor emite su primer mensaje
    4. El receptor emite su segundo y último mensaje
    5. El emisor emite su segundo y último mensaje

vi. El emisor desconecta la llamada apretando el botón “Finalizar llamada”

vii. En caso de no tener que hacer otra llamada, el emisor deja el teléfono, caso contrario, el proceso continúa en el punto “iii”.

4. Finalmente, para asegurarnos de que nuestro algoritmo cumple con todos los requerimientos solicitados, lo pondremos a prueba realizando una validación del proceso, al cuestionarnos nuestra propia solución a través de casos de prueba. Planteamos los casos a continuación:
   a. ¿Se cumple con las premisas iniciales (que la llamada la pueda realizar cualquier persona y no alguien en particular, que se pueda llamar a cualquier número de 8 dígitos, etc.)? La respuesta es SÍ.
   b. ¿El proceso funciona correctamente si el emisor al momento de ingresar alguno de los 8 dígitos se equivoca e ingresa un carácter extraño, como el “*”? La respuesta es NO. Para resolver este problema que encontramos, agregaremos un nuevo paso en el algoritmo, concretamente entre los puntos “iv” y “v” llamado “iv.1” el cual podría ser:

      iv1. Si se marcó incorrectamente el número, el algoritmo continúa en el punto “vii”.

      Con esta salvedad, damos por corregido el algoritmo.

Como reflexión final, podemos decir que la lógica de la programación, o como llamamos en este curso: la “forma de pensar del programador”, se trata de una serie de habilidades que permite plantear una solución a un problema dado, utilizando el razonamiento y una forma de pensar sistemática y ordenada, que nos permite llegar a la formulación de algoritmos abstractos en un nivel de detalle determinado y que es posible implementarlos con un lenguaje de programación en particular.

5. ANEXO - Otros algoritmos resueltos

En los algoritmos resueltos que vemos a continuación, se presenta el planteo final directo, salteando el mecanismo de resolución en tres pasos, directo a la solución.

ALGORITMO A - Encendido de TV

Precondiciones:

1 - El actor ("Persona") debe tener las facultades físicomotrices y psicológicas necesarias para realizar la acción.

2 - La TV debe estar en condiciones para ser encendida (energía eléctrica, cableado).

3 - El actor debe tener los medios para encender la TV (aplica solo para TVs que se encienden con únicamente control remoto)

Desarrollo:

1 - El actor se acerca a la tv.

2 - El actor toma el control remoto.

3 - El actor pulsa el botón de encendido/apagado.

3.1 - Si no enciende.

3.1.1 - Se repite desde el punto 3, un máximo de 7 veces.

3.2 - Si enciende.

3.2.1 - El actor sonríe satisfecho .

4 - Fin del algoritmo.

ALGORITMO B - Calentar agua para preparar un té

Precondiciones:

1 - El actor ("Persona") debe tener las facultades físicomotoras y psicológicas necesarias.

2 - El actor debe tener los medios para preparar el té (Aplica solo para agua calentada en un pava sobre hornalla a gas).

Desarrollo:

1 - El actor se acerca a la cocina.

2 - El actor toma la pava.

3 - El actor se dirige a la canilla.

4 - El actor abre la canilla

5 - El actor llena la pava con agua.

6 - El actor se dirige a la hornalla.

7 - El actor apoya la pava en la hornalla.

8 - El actor toma los fósforos/encendedor.

9 - El actor enciende un fósforo/encendedor.

10 - El actor abre la llave de gas de la hornalla.

10.1 - Si no sale gas.

10.1.1 - El actor revisa la llave de paso del gas.

10.1.1.1 - Si no funciona la llave.

10.1.1.1.1 - Se repite desde el paso 10.1.1, máximo 3 veces.

10.1.1.2 - Si funciona la llave y sale gas.

10.1.1.2.1 - Continua el Algoritmo.

10.2 - Si sale gas.

10.2.1 - Continua el Algoritmo.

11 - El actor posiciona el fósforo/encendedor encendido en la hornalla encendida.

11.1 - Si no enciende el fuego.

11.1.1 - Repetir desde el paso 10, máximo 4 veces o hasta que se quede sin fósforos/encendedor.

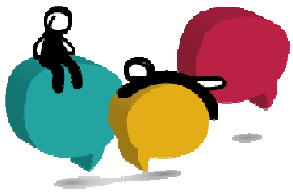
11.2 - si enciende el fuego.

11.2.1 - Continua el Algoritmo.

12 - El actor espera a que el agua levante la temperatura necesaria.

13 - El actor apaga la hornalla.

14 - Fin del algoritmo.



Actividades de la Unidad 3:

- Consignas:
  - Punto 1: Según lo visto en los materiales. describa los elementos principales (físicos o no) que cree necesario que una persona debe tener o saber a la hora de programar
  - Punto 2: Investigue y resuma en una breve reseña la evolución de los distintos lenguajes de programación que derivaron en los actuales Java y C#
  - Punto 3: Utilizando lenguaje natural, describa un algoritmo para encender un auto. Aclare todas las precondiciones y postcondiciones que crea necesarias.

Pueden crear un documento en Word (o similar, compatible con formato doc) y subirlo con las 3 preguntas en el link disponible en la Unidad 3.

- **¡Atención! Esta actividad no se debe subir a los foros públicos, sino deberá ser entregada en el lugar especialmente identificado para este caso.**

- **¡Atención! En caso de no hacer la entrega a tiempo, como se estipula en los mensajes y foros del Campus virtual, NO se debe realizar la entrega en los foros públicos. Las fechas y plazos son los mismos para todos y estos deberán respetarse.**

Lo que vimos

- Principales aspectos necesarios relacionados a la "forma de pensar" del programador
- Prácticas de razonamiento y resolución de problema necesarias para desarrollar algoritmos
- Proceso para el planteo de soluciones a un problema dado
- Tipos de algoritmos y la forma de representarlos
- Caso integrador de los temas vistos

Lo que viene:

- Formas principales de representación de algoritmos que serán utilizados en este Módulo: diagramas de flujo y seudocódigo
- Convenciones de los mismos
- Formatos de los diagramas de flujo y que los comprendan a través de un ejemplo práctico
- Aspectos principales de la representación de algoritmos usando seudocódigo