Propuesta de planificación anual para Tecnologías de la Información, 3º año de la NES (TI3), CABA

Versión del 27 de marzo de 2017[1]

[1] Esta obra está bajo Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Índice

Índice 2

Historial de versiones 6

Prólogo 7

Planificación TI3 9

Clase 1 - Introducción 9
Objetivos 9
Contenido y desarrollo 9
Herramientas 9
Clase 2 - Noción de programa y proyecto simple I 9
Objetivos 9
Contenido y desarrollo 10
Herramientas 10
Clase 3 - Métodos I 10
Objetivos 10
Contenido y desarrollo 10
Herramientas 11
Clase 4 - Alternativa condicional 11
Objetivos 11
Contenido y desarrollo 11
Herramientas 11
Clase 5 - Proyecto simple II 11
Objetivos 11
Contenido y desarrollo 12
Herramientas 12
Clase 6 - Variables I 12
Objetivos 12
Contenido y desarrollo 12
Herramientas 1
Clase 7 - Diferencia entre programa y algoritmo 13
Objetivos 13
Contenido y desarrollo 13
Herramientas 13
Clase 8 - Lenguajes de programación 13
Objetivos 13
Contenido y desarrollo 13
Herramientas 13
Clase 9 - El lenguaje de máquina 14

Objetivos 14
Contenido y desarrollo 14
Herramientas 14
Clase 10 - Arquitectura básica de una computadora 15
Objetivos 15
Contenido y desarrollo 15
Herramientas 15
Clase 11 - Almacenamiento y memoria I 15
Objetivos 15
Contenido y desarrollo 16
Herramientas 16
Clase 12 - Almacenamiento y memoria II 16
Objetivos 16
Contenido y desarrollo 17
Herramientas 17
Clase 13 - CPU I 17
Objetivos 17
Contenido y desarrollo 17
Herramientas 18
Clase 14 - CPU II 18
Objetivos 18
Contenido y desarrollo 18
Herramientas 19
Clase 15 - Relación hardware-software 19
Objetivos 19
Contenido y desarrollo 19
Herramientas 19
Clase 16 - Sistemas operativos I 20
Objetivos 20
Contenido y desarrollo 20
Herramientas 20
Clase 17 - Sistemas operativos II 20
Objetivos 20
Contenido y desarrollo 20
Herramientas 21
Clase 18 - Sistemas operativos III 21
Objetivos 21
Contenido y desarrollo 21
Herramientas 21
Clase 19 - Software libre 21
Objetivos 21
Contenido y desarrollo 22
Herramientas 22

Clase 20 - Software malicioso 22
Objetivos 22
Contenido y desarrollo 22
Herramientas 23
Clase 21 - Repaso de programación 23
Objetivos 23
Contenido y desarrollo 23
Herramientas 23
Clase 22 - Métodos II 23
Objetivos 23
Contenido y desarrollo 24
Herramientas 24
Clase 23 - Proyecto integrador I 24
Objetivos 24
Contenido y desarrollo 24
Herramientas 24
Clase 24 - Métodos III 25
Objetivos 25
Contenido y desarrollo 25
Herramientas 25
Clase 25 - Repetición simple 25
Objetivos 25
Contenido y desarrollo 25
Herramientas 26
Clase 26 - Variables II 26
Objetivos 26
Contenido y desarrollo 26
Herramientas 26
Clase 27 - Repetición condicional 27
Objetivos 27
Contenido y desarrollo 27
Herramientas 27
Clase 28 - Proyecto integrador II 28
Objetivos 28
Contenido y desarrollo 28
Herramientas 28
Clase 29 - Proyecto integrador III 28
Objetivos 28
Contenido y desarrollo 28
Herramientas 28

Secuencia de actividades 29

Clase 1 - Introducción 29

Clase 2 - Noción de programa y proyecto simple I 30
Clase 3 - Métodos I 31
Clase 4 - Alternativa condicional 33
Clase 5 - Proyecto simple II 37
Clase 6 - Variables I 38
Clase 7 - Diferencia entre programa y algoritmo 40
Clase 8 - Lenguajes de programación 43
Clase 9 - El lenguaje de máquina 47
Clase 10 - Arquitectura básica de una computadora 51
Clase 11 - Almacenamiento y memoria I 57
Clase 12 - Almacenamiento y memoria II 61
Clase 13 - CPU I 67
Clase 14 - CPU II 68
Clase 15 - Relación hardware-software 72
Clase 16 - Sistemas operativos I 75
Clase 17 - Sistemas operativos II 81
Clase 18 - Sistemas operativos III 86
Clase 19 - Software libre 89
Clase 20 - Software malicioso 99
Clase 21 - Repaso de programación 106
Clase 22 - Métodos II 107
Clase 23 - Proyecto integrador I 108
Clase 24 - Métodos III 109
Clase 25 - Repetición simple 110
Clase 26 - Variables II 111
Clase 27 - Repetición condicional 112
Clase 28 - Proyecto integrador II 113
Clase 29 - Proyecto integrador III 114

Bibliografía **115**
Material didáctico 115
Material ampliatorio 115
Software 115

Créditos **116**
Autores 116
Diseño gráfico e ilustración 116
Coordinación Iniciativa Program.AR 116
Autoridades Fundación Dr. Manuel Sadosky 116

Historial de versiones

Versión	Cambios
15/02/2017	Versión inicial con la planificación completa y secuencias didácticas de las clases 1 a 8.
24/02/2017	Se agregaron las secuencias didácticas de las clases 9 a 19.
03/03/2017	Se agregaron las secuencias didácticas de las clases 20 a 29. Se realizaron pequeñas correcciones en las secuencias didácticas de las clases 9 a 19.
27/03/2017	Se mejoraron las descripciones de objetivos y contenidos de las clases para reflejar las estrategias didácticas de cada una. Se corrigieron aspectos menores de diseño. Algunas clases que eran muy largas se adecuaron mejor al tiempo disponible.

Propuesta de planificación anual para Tecnologías de la Información, 3º año de la NES (TI3), CABA / Teresa Alberto ... [et al.] ; ilustrado por Jaqueline Schaab. - 1a edición para el profesor - Ciudad Autónoma de Buenos Aires : Fundación Sadosky, 2017.
Libro digital, PDF - (Program.AR. Materiales para Tecnologías de la Información, CABA ; 2)

Archivo Digital: descarga y online
ISBN 978-987-27416-3-1

1. Computación. 2. Docente de Secundaria. 3. Capacitación docente. I. Alberto, Teresa II. Schaab, Jaqueline, ilus.
CDD 371.1

Prólogo

Es complejo entender el mundo en el que vivimos sin cierto conocimiento acerca de cómo funcionan las computadoras y los programas que las comandan. Interactuamos con ellas todo el tiempo: cuando buscamos en Internet, cuando chateamos mediante nuestros celulares y también cuando pagamos el boleto en el transporte público, o cuando mediante el dispositivo conocido como "pedal" activamos el sistema de frenado de un auto. La tecnología permea nuestra vidas y las discusiones que la rodean van ganando lugar en la agenda pública. Si no conocemos de estos temas, nuestra posibilidad de participar como ciudadanas y ciudadanos plenos se ve reducida. Por eso es tan importante que la escuela aborde las Ciencias de la Computación y en consecuencia la importancia de la materia Tecnologías de la Información.

Ciencias de la Computación es el nombre que recibe la disciplina que estudia cómo funcionan las computadoras y los programas que las controlan, cómo se comunican éstas entre sí, y tantas otras cuestiones relacionadas con la programación, la construcción de sistemas y el procesamiento de información de todo tipo.

La materia **Tecnologías de la Información** es una de las formas en que la escuela argentina empieza a responder a la necesidad de enseñar Ciencias de la Computación, y de ahí su importancia. La Iniciativa Program.AR viene trabajando desde hace varios años para contribuir con la llegada de las Ciencias de la Computación a la escuela, y por ende desea contribuir a que esta materia sea un éxito. Por ese motivo, ha puesto su equipo de profesionales a trabajar en una propuesta de trabajo para el aula. Esta primera edición está pensada para "TI3", es decir, la materia Tecnologías de la Información de tercer año de la Nueva Escuela Secundaria de la Ciudad Autónoma de Buenos Aires.

Se trata de una **planificación de todo el año y sus correspondientes secuencias didácticas**, ajustadas al plan de estudio oficial de la materia, con el objetivo de brindarle al docente un marco sólido para tomar de base a la hora de pensar sus propias clases. En ellas se abordan temas de programación, organización y arquitectura de computadoras, sistemas operativos y seguridad informática.

Propone un **recorrido** original: comienza presentando nociones elementales de programación en una plataforma gráfica, apta para adolescentes sin ningún conocimiento previo, con el objetivo de explicar la noción de programa y una introducción a la de algoritmo. El paso siguiente es hacer algunos ejercicios muy sencillos en un lenguaje textual, para poder preguntarse qué contienen los programas ejecutables, aquellos a los que en nuestra experiencia cotidiana simplemente les hacemos doble clic. Esa pregunta sirve de puntapié inicial para indagar sobre cómo funcionan las computadoras (su organización y su arquitectura), lo que nos permite transformar una noción cercana a una caja opaca, casi mágica, en un concepto sobre el que podemos razonar.

Luego de esa "sumergida" en profundidad, volvemos a acercarnos a la "superficie" pasando por los conceptos de sistema operativo, de virus y de algunos otros relacionados con los aspectos de la seguridad informática que tienen contacto más directo con nuestro uso cotidiano de las computadoras. Terminamos el año volviendo a la programación en entornos para principiantes, repasando y profundizando las nociones ya adquiridas y visitando algunas nuevas.

Este trabajo se pone a disposición de la comunidad bajo una **licencia *Creative Commons***[2] como una forma de incentivar la creación de obras derivadas. Dicho de otra forma, fomentamos activamente que las y los colegas generen sus propias versiones de este material y las compartan con la comunidad.

Consideramos que este material se encuentra en la categoría **trabajo en curso**, y que por lo tanto irá mejorando a medida que vaya siendo usado y discutido. Esperamos que las y los colegas lo encuentren útil y quedamos atentos a críticas o comentarios. Los esperamos en info@program.ar y los invitamos a revisar periódicamente nuestro sitio web, o seguirnos en las redes sociales, para mantenerse al tanto de las futuras versiones.

El equipo de Program.AR

---

www.program.ar

@Programar2020 | programar2020

[2] Específicamente, una licencia "Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional", cuyos detalles pueden consultarse en https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es.

Planificación TI3

Clase 1 - Introducción

Objetivos
• Aplicar estrategias y técnicas para crear animaciones y videojuegos mediante entornos de programación educativa. • Comprender el carácter específico de las instrucciones empleadas en programación. • Realizar una primera aproximación a conceptos centrales para la realización de un programa.
Contenido y desarrollo
En esta clase se presentará la materia e instalará y presentará la herramienta Alice. Los componentes de Alice se analizarán como si fueran las partes necesarias para la realización de una película: la cámara, la iluminación, la locación, la escenografía, los actores y el guión. Se relacionará el comportamiento de los personajes/actores durante la ejecución con las instrucciones indicadas en el guión. Ver Actividades.
Herramientas
Alice

Clase 2 - Noción de programa y proyecto simple I

Objetivos
• Comprender la noción de programa y la diferencia entre tiempo de creación y de ejecución del mismo. • Establecer relaciones entre el orden de las acciones programadas y los resultados obtenidos. • Asumir posturas activas para la exploración de las herramientas, opciones y limitaciones que propone un entorno.

Aplicar los conceptos adquiridos para desarrollar animaciones simples. Planificar y realizar un proyecto grupal de animación y/o juego que permita abordar alguna temática significativa para el grupo.
Contenido y desarrollo
Se pedirá a las y los estudiantes que realicen una animación simple para indagar sobre los métodos disponibles en Alice, intercambiando el orden de las instrucciones y analizando los resultados al ejecutar el guión. Se presentarán las nociones de programa y secuencialidad. Para finalizar, en esta clase se comenzará un proyecto grupal que consistirá en la realización de una animación y que se trabajará a lo largo del trimestre. Ver Actividades.
Herramientas
Alice

Clase 3 - Métodos I

Objetivos
• Introducir la idea de abstracción a través del uso de métodos en Alice. • Reconocer cómo descomponer el problema/guión en acciones abstractas más abarcativas como estrategia para encarar la creación de un programa. • Reconocer la función de los métodos.
Contenido y desarrollo
A partir del relato de una historia se pedirá a las y los estudiantes que programen un guión. Debido a la longitud y repetición de acciones en la historia, la creación del programa se volverá tedioso y extenso. A posteriori, se introducirá como estrategia de resolución la identificación de acciones abarcativas que nuclean secuencias de instrucciones más simples dentro de la historia. Se identificarán esas acciones de manera conjunta y se procederá a reescribir el programa aprovechando el recurso de los métodos para plasmar las acciones más complejas y abstractas que componen la historia. Se identificarán las ventajas del uso de métodos al comparar los dos programas construidos, poniendo énfasis en la legibilidad, la extensión y claridad del programa resultante. Ver Actividades.

Herramientas
Alice; escenario inicial y guión de la actividad.

Clase 4 - Alternativa condicional

Objetivos
• Comprender el concepto de alternativa condicional y de flujo de control alternativo o no secuencial. • Utilizar correctamente los comandos necesarios para ejecutar alternativas condicionales. • Realizar programas adecuados frente a escenarios cambiantes. Identificar el uso de alternativas condicionales tanto en situaciones cotidianas como en juegos y aplicaciones conocidas. • Distinguir entre la programación de animaciones y la programación de juegos con interacción del usuario a partir del uso de eventos.
Contenido y desarrollo
Uso de condicionales. Interactividad. El objetivo es que las y los estudiantes programen un juego en el que un personaje prenda todas las luces de una fila de casilleros, teniendo en cuenta que en ellos puede o no haber una luz. Este ejercicio servirá para introducir el concepto de alternativa condicional a partir de la instrucción if/else de Alice. Se buscarán ejemplos de toma de decisiones en la vida cotidiana, en juegos y aplicaciones para explicar el uso de alternativas condicionales. Ver Actividades.
Herramientas
Alice y el escenario inicial de la actividad.

Clase 5 - Proyecto simple II

Objetivos

• Aplicar técnicas y estrategias para crear animaciones y videojuegos mediante entornos de programación educativos. • Propiciar trabajo activo y colaborativo entre pares.
Contenido y desarrollo
Se trabajará sobre el proyecto presentado en la clase 2. Se presentará el nivel de avance de cada grupo y los obstáculos y objetivos a futuro. A partir de esto, se realizará un intercambio para abordar de manera conjunta los aspectos de la herramienta que requiera cada pieza. Ver Actividades.
Herramientas
Alice

Clase 6 - Variables I

Objetivos
• Seleccionar y aplicar estrategias para la resolución de un problema simple. • Comprender el concepto de variable y relacionarlo con la memoria y las formas de almacenamiento de las computadoras.
Contenido y desarrollo
Se discutirá la diferencia entre juego y animación y se mostrará cómo usar Alice para hacer programas interactivos. Como primer ejemplo se puede hacer un programa en el que el usuario ingrese su nombre, y algún personaje lo diga. Como ejercicio para introducir variables, se solicitará que ingresando una única vez el nombre del usuario varios personajes lo saluden. Mediante una dramatización con cajas donde se guardarán papeles con el nombre de un estudiante se ilustrará el uso y el concepto de variable como un lugar donde se puede almacenar un dato y después recuperar esa información que está guardada. Ver Actividades.
Herramientas
Alice

Clase 7 - Diferencia entre programa y algoritmo

Objetivos
• Reconocer la función de los algoritmos y su representación en pseudocódigo.
Contenido y desarrollo
Se repasarán los conceptos de programación trabajados hasta el momento. Luego, se realizará un ejercicio en Alice en parejas en el que las y los estudiantes deberán ingresar los puntajes de dos equipos, compararlos y decidir si hubo un ganador o un empate. Tras analizar entre toda la clase las soluciones propuestas se presentará el concepto de algoritmo, se trabajará en la resolución de otros problemas simples, la escritura de pseudocódigos, la posibilidad de diversas estrategias de resolución y la diferencia entre un programa y un algoritmo. Ver Actividades.
Herramientas
Alice

Clase 8 - Lenguajes de programación

Objetivos
• Reconocer diferencias y similitudes entre lenguajes y entornos de programación. • Reforzar la idea de que un mismo algoritmo puede tener varias implementaciones, incluso en varios lenguajes.
Contenido y desarrollo
Se presentará un problema para resolver con un algoritmo pensado entre toda la clase y se programará, también colectivamente, en lenguaje C simplificado para introducir su sintaxis, explorar las principales diferencias con Alice y resaltar que hay similitudes independientemente del lenguaje elegido para la implementación. Ver Actividades.

Herramientas
Lenguaje C y compilador de C (ya instalado en Huayra). Archivos disponibles en https://goo.gl/hqZRj0.

Clase 9 - El lenguaje de máquina

Objetivos
• Reflexionar sobre la relación entre lenguajes de programación de alto y bajo nivel, y la necesidad de los compiladores. • Distinguir la diferencia entre código fuente y código binario (también llamado ejecutable). • Analizar la función, estructura y funcionamiento de los lenguajes de bajo nivel.
Contenido y desarrollo
La clase requerirá el uso de Huayra (u otra distribución de Linux) y la instalación de un programa gratuito de desarrollo argentino denominado BARF. El/la docente distribuirá el software para su instalación al comienzo de la clase. Luego, utilizará el programa en C realizado en la clase 8 y mostrará que durante su uso se generó un nuevo archivo. Se analizará este archivo, su tamaño y cambios al realizarle modificaciones al programa original escrito en C. Esto servirá de introducción para presentar el lenguaje máquina, el proceso de compilación y la diferencia entre código fuente y ejecutable. Mediante el uso de BARF se podrá visualizar el lenguaje máquina generado al compilar el programa en C. Entre toda la clase se analizará el gráfico generado por BARF al *desensamblar* el programa en C, lo que servirá para comprender mejor algunos detalles del lenguaje máquina. Se propondrá como actividad que las y los estudiantes modifiquen el programa en C y describan la relación entre sus modificaciones y la imagen generada por BARF. Para finalizar se realizará un breve repaso de los nuevos conceptos trabajados en clase. Ver Actividades.
Herramientas
C y BARF

Clase 10 - Arquitectura básica de una computadora

Objetivos
• Comprender qué características determinan que un objeto sea una computadora. • Identificar qué partes componen una computadora. • Establecer relaciones entre los distintos componentes. • Deducir las funciones principales del CPU y la memoria RAM.
Contenido y desarrollo
Como primera actividad se deberá decidir qué objetos son una computadora entre un conjunto de diversos dispositivos electrónicos. A continuación se buscará identificar los componentes que forman parte de la computadora: el procesador, la memoria RAM y distintos periféricos de entrada, salida y entrada/salida. En paralelo se irá construyendo un gráfico que permita visualizar cómo se relacionan todos estos componentes. Vinculando la primera actividad con la segunda, se reconocerán los componentes mencionados en la actividad anterior, en diversos dispositivos (computadora, consola, celular, etc.). Por último, se intentará deducir cuáles son las funciones principales de la memoria RAM y el procesador a partir de lo que las y los estudiantes ya saben, y de pensar computadoras sin alguno de estos componentes. Ver Actividades.
Herramientas
Partes de *hardware* en vivo o, en su defecto, video y/o fotos.

Clase 11 - Almacenamiento y memoria I

Objetivos
• Diferenciar entre 2 tipos de memoria: la volátil y la persistente. • Comprender la magnitud de las unidades de almacenamiento. • Comprender la noción de tiempo de acceso. • Reconocer las características de la jerarquía de memoria.

- Vincular las nociones de variable y programa con la de memoria.
- Reconocer la necesidad de direcciones de memoria.

Contenido y desarrollo

Como primer acercamiento al tema se indagará acerca de cuáles son los conceptos y conocimientos previos que las y los estudiantes tienen acerca del mismo.

Luego, se plantearán una serie de preguntas que motiven la distinción entre lo que se conoce como memoria volátil y memoria persistente.

En la siguiente actividad, se pretenderá comprender la magnitud de las distintas unidades utilizadas para almacenar datos en una computadora (Byte, KB, MB, GB, TB, PB, etc.). Para ello se propondrá realizar ciertas modificaciones en un archivo originalmente en blanco y notar cómo cambia el tamaño de dicho archivo.

A continuación, el/la docente explicará el concepto de tiempo de acceso y recorrerá junto con sus alumnas y alumnos un gráfico que muestre la jerarquía de memoria, identificando las características más sobresalientes: tiempo de acceso, costo y tamaño.

Retomando el concepto de variable y de programa, las y los estudiantes deberán decidir y argumentar en qué tipo de memoria (RAM o disco) se guardan distintos recursos.

Se problematizará cómo hace la memoria para saber dónde guardó cada dato, surgiendo la noción de dirección de memoria.

Por último, deberán indagar sobre las características de la memoria RAM y el disco de los dispositivos con los que cuenten en clase (celulares, computadoras, tabletas, etc.).

Ver Actividades.

Herramientas

-

Clase 12 - Almacenamiento y memoria II

Objetivos

- Comprender el concepto de caché y qué problema resuelve: el acceso rápido a la información frecuente.

- Conocer cómo se puede almacenar y liberar un dato de caché.

Contenido y desarrollo

Para motivar la necesidad de tener una memoria *caché*, se realizará una primera actividad en la que habrá que calcular el tiempo que toma ejecutar un programa que lee y escribe en una memoria USB o pendrive. La idea es comparar distintos escenarios en donde varíe el tiempo de acceso de la memoria y se note el contraste respecto del tiempo que le toma ejecutar una instrucción a la CPU.

A continuación, se propone una discusión con la clase para pensar cómo se podría solucionar este problema, buscando acercarse a la idea de memoria *caché*.

Se comentará cuáles son los pasos que se siguen para guardar los datos en *caché* y se reflexionarán cuáles son las ventajas y desventajas de tener memoria *caché*.

Por último, se comentará uno de los algoritmos de reemplazo de datos en *caché*, LRU.

Ver Actividades.

Herramientas

-

Clase 13 - CPU I

Objetivos

- Descubrir las características de los procesadores que poseen las y los estudiantes en sus dispositivos.
- Comprender cómo se mide la velocidad de un procesador.
- Comprender el significado de la unidad de medida GHz.

Contenido y desarrollo

Como primera actividad, las y los estudiantes deberán descubrir distintas características del procesador que tienen en sus dispositivos y se debatirán distintos aspectos acerca de los mismos.

A continuación se problematizará el significado del término "GHz", recapitulando lo visto en la clase 9.

Por último, se buscará que las y los estudiantes lleguen a la conclusión de que la velocidad del procesador es una de las medidas que nos permite comparar qué tan veloces son dos computadoras.

Ver Actividades.

Herramientas

-

Clase 14 - CPU II

Objetivos

- Comprender cómo funciona la CPU.
- Establecer la relación entre CPU y memoria RAM.
- Conocer cómo se compone por dentro una CPU.
- Comprender cómo ejecuta un programa la CPU.

Contenido y desarrollo

Como actividad introductoria se reconstruirá el gráfico realizado en la clase 10 pero haciendo énfasis en la relación entre CPU y memoria RAM.

Luego se buscará fomentar un debate, a través de preguntas que realice el/la docente, que permita ensayar distintas propuestas acerca de cómo podría funcionar una CPU.

Al finalizar el debate, se reflexionará acerca de cuáles son los pasos que realiza la CPU para ejecutar un programa, usando como ejemplo un pequeño programa en lenguaje ensamblador. Esta actividad dará pie a la construcción de un gráfico que muestre por dentro la CPU, destacando aquellos componentes más importantes: registros, Unidad de Control y ALU.

Para afianzar cómo una CPU ejecuta un programa, se sugiere repasar los pasos utilizando el diagrama y determinando cuál de los componentes realiza cada tarea.

Como actividad entregable, se pedirá que escriban todos los pasos que se realizan para ejecutar un programa de una instrucción.

Por último, se mostrará un procesador real y se concluirá que "una computadora no es más que la interconexión de muchos componentes electrónicos por los que circula electricidad", siendo todo lo anterior distintas capas de abstracción. Ver Actividades.
Herramientas
-

Clase 15 - Relación hardware-software

Objetivos
• Adquirir una perspectiva general del funcionamiento de una computadora. • Vincular el hardware y el software entre sí considerando todos los componentes trabajados anteriormente.
Contenido y desarrollo
Como actividad de repaso, se propiciará una breve discusión que retome varios de los conceptos vistos hasta el momento: qué es el hardware, qué es el software, cuáles son los componentes de hardware presentes en una computadora y qué hacen, etc. Luego, se presentará un trabajo práctico para resolver en clase en el que se propone un análisis para abordar la mayoría de los temas de las clases previas. El trabajo será grupal y escrito. Se presentará un programa y se deberá describir su funcionamiento y analizar las limitaciones e inconvenientes si se lo intentara ejecutar en una computadora sin algunos de sus componentes (sin CPU, con memoria RAM, sin memoria *caché*, etc.). Ver Actividades.
Herramientas
-

Clase 16 - Sistemas operativos I

Objetivos
• Descubrir que entre el hardware y las aplicaciones que usamos existe una capa intermedia: el sistema operativo. • Comprender la utilidad y funcionalidades de los sistemas operativos. • Comprender cuáles son sus partes más importantes.
Contenido y desarrollo
Se retomará el programa C que las y los estudiantes hicieron en la clase 8 y se lo compilará en dos sistemas operativos distintos para compararlos y determinar que no son iguales a pesar de que ni el código del programa original ni la máquina usada cambiaron. A partir de esa observación se motivará una introducción a los sistemas operativos como mediadores entre la parte física de la computadora y el software que usamos. Se distinguirá entre las funciones básicas del sistema operativo, es decir, el kernel y la interfaz de usuario. Se analizará el origen y las funciones de los drivers o controladores. Ver Actividades.
Herramientas
Huayra Linux, Windows, MinGW y herramientas propias de ambos sistemas operativos.

Clase 17 - Sistemas operativos II

Objetivos
• Comprender algunas funciones más avanzadas de los sistemas operativos, en particular referidas al manejo de recursos, procesos, usuarios y el sistema de permisos.
Contenido y desarrollo
Se profundizará la idea de sistema operativo como un programa que funciona como nexo entre el hardware y el resto de los programas a través del análisis de los procesos en ejecución y los monitores de recursos.

Se trabajará sobre el sistema de permisos y la creación de nuevos usuarios. Ver Actividades.
Herramientas
Huayra Linux, Windows, Android y herramientas propias de dichos sistemas operativos.

Clase 18 - Sistemas operativos III

Objetivos
• Comprender el origen de las actualizaciones y aplicaciones de un sistema operativo. • Analizar criterios para una elección adecuada de sistema operativo.
Contenido y desarrollo
Se analizarán los objetivos de las actualizaciones de los sistemas operativos, presentando los sistemas de paquetes. Se discutirá qué características o prestaciones de un sistema operativo analizar para poder decidir, entre distintas opciones, cuál se adecúa mejor a nuestras necesidades. Por último, se reflexionará acerca de qué pasaría si la computadora que usamos no tuviera instalado un sistema operativo. Ver Actividades.
Herramientas
-

Clase 19 - Software libre

Objetivos

- Comprender las características del software libre y propietario.
- Distinguir entre software libre y gratuito.
- Familiarizarse con el concepto de copyleft.

Contenido y desarrollo

Como primera actividad se propondrá realizar una reflexión sobre el acceso y los derechos de uso de bienes culturales que hayan obtenido recientemente (libros, canciones, juegos, etc.).

Luego se avanzará sobre las nociones de software libre y propietario, considerando distintos programas de uso cotidiano. Se distinguirá entre que un software sea libre y que sea gratuito. A través de distintos ejemplos se buscará comprender las libertades del usuario y las implicancias de acceso al código fuente, confluyendo, hacia el final de la clase, al concepto de copyleft.

Por último, se presentará una actividad para relacionar las características del software con licencia copyleft y con licencia propietaria.

Ver Actividades.

Herramientas

-

Clase 20 - Software malicioso

Objetivos

- Aproximarse al concepto de vulnerabilidad.
- Comprender qué es el software malicioso, cómo funciona y cuál es su origen.
- Favorecer el uso responsable de las tecnologías de la información y la comunicación.

Contenido y desarrollo

Se realizará una aproximación al concepto de vulnerabilidad para introducir luego el de software malicioso. Para ello se analizará el código de un programa que tiene una falla de seguridad.

A continuación se realizará un trabajo grupal sobre las características de distintos tipos de programas maliciosos.

Otra de las nociones que se estudiará es la de phishing a través del análisis de escenarios maliciosos y seguros. ¿Cómo podemos distinguirlos?

La actividad de cierre consistirá en reflexionar cómo podemos proteger nuestros sistemas y nuestros datos frente a este tipo de amenazas. Ver Actividades.
Herramientas
Caja de herramientas

Clase 21 - Repaso de programación

Objetivos
• Repasar los conceptos de programación trabajados en la primera parte de la materia.
Contenido y desarrollo
Se programará un juego que combine los temas de programación vistos en las clases iniciales. El objetivo es retomar la idea de programa entendido como guión, la modificación del orden secuencial de sus instrucciones a partir de alternativas condicionales y la interacción con el usuario en el transcurso de la ejecución de un programa, que brinda información que será guardada en una variable en memoria para su uso posterior y permitirá trabajar con información desconocida y cambiante al momento de crear el programa. Ver Actividades.
Herramientas
Alice

Clase 22 - Métodos II

Objetivos
• Profundizar la comprensión del uso de métodos como estrategia de abstracción.

Contenido y desarrollo
Se volverá a trabajar con la animación “Encuentro de tercer tipo”, pero bajo el formato de un juego del tipo *Elige tu propia aventura*. Para ello se eliminará el final de la historia y se lo dejará en suspenso a la espera de la decisión del usuario, a quien se le presentarán dos alternativas de desenlace. A partir del trabajo con un juego más complejo se buscará que las y los estudiantes vuelvan sobre los conceptos anteriores, para recuperar su significado y apropiarse de los recursos aprendidos. Se trabajará nuevamente sobre la lectura de una historia por lo que se espera que se recupere nuevamente el proceso de trabajo para la creación de un programa: la lectura del guión, la identificación de métodos o acciones más abarcativas con los que organizar el accionar de los personajes, y la implementación del programa con Alice. Ver Actividades.
Herramientas
Alice y guión de la actividad.

Clase 23 - Proyecto integrador I

Objetivos
• Sintetizar y combinar los conceptos de programación abordados en las clases previas.
Contenido y desarrollo
Se delinearán las pautas iniciales para la programación de un juego en Alice por parte de las y los estudiantes. El proyecto será grupal y se irá complejizando con nuevas consignas y recursos a medida que se avance con los contenidos de programación en las sucesivas clases. Ver Actividades.
Herramientas
Alice

Clase 24 - Métodos III

Objetivos
• Profundizar la estrategia de abstracción que facilitan los métodos, la pertinencia de su creación y la definición de un nombre adecuado.
Contenido y desarrollo
Se les presentará a las y los estudiantes varios programas como posibles soluciones a un problema dado. La propuesta es que, en grupos, elijan la abstracción más adecuada que resuelva el enunciado. Se les pedirá que dentro de cada grupo debatan cuál de ellas representa la solución más adecuada al problema y que justifiquen por qué. Ver Actividades.
Herramientas
Alice y archivos disponibles en https://goo.gl/5d4HWe.

Clase 25 - Repetición simple

Objetivos
• Repasar el concepto de secuencialidad y de alternativas condicionales como dos tipos de *recorridos* de las instrucciones dentro de un programa. • Introducir las estructuras de repetición simple, que modifican el flujo secuencial del programa y repiten las instrucciones una cantidad fija de veces.
Contenido y desarrollo
Se retomará la actividad de la clase 21 y se pedirá a las y los alumnos que programen el juego limitando a tres intentos las oportunidades del jugador. Luego de que los y las estudiantes exploren distintas soluciones se complejizará la consigna y tras unos minutos de trabajo se propiciará un breve debate para discutir la idea de secuencialidad de un programa y revisar la alternativa condicional desde esa perspectiva. Se

resaltará la capacidad del condicional de alterar el flujo secuencial de un programa para que no transcurra de manera lineal. Se continuará con la ruptura de la secuencialidad con otro tipo de recursos como la repetición simple y se presentará el uso de “loop” o “ciclo” de Alice que las y los alumnos deberán incluir en su solución. Para finalizar se discutirá sobre acciones que se repiten un número finito de veces en la vida cotidiana. Ver Actividades.
Herramientas
Alice

Clase 26 - Variables II

Objetivos
• Profundizar el funcionamiento de las variables como contadores de una cantidad dentro de la dinámica de un juego. • Reforzar la idea de que una variable permite memorizar un valor que puede cambiar a lo largo de la ejecución del programa.
Contenido y desarrollo
Se instalará un escenario de Alice con un juego simple en las computadoras de las y los estudiantes y se les pedirá que le agreguen un contador de puntaje que comience en cero y se incremente o decremente en función del accionar del jugador. Se profundizará el funcionamiento de las variables, con un uso no explorado: utilizarlas para contar una cantidad. Luego se complejizará el juego y las y los estudiantes deberán incluir más oportunidades para el jugador y acumular el puntaje de las mismas. Se hará incapié en la idea de que una variable permite memorizar un valor que puede cambiar a lo largo de la ejecución del programa. Se avanzará con el proyecto integrador del Juego, definiendo los últimos detalles y resolviendo las dudas que existan. Ver Actividades.

Herramientas
Alice y escenario inicial Juego de feria.

Clase 27 - Repetición condicional

Objetivos
• Introducir la repetición condicional (el “while” o “repetir mientras que…”), como una forma más avanzada de romper la secuencialidad de un programa. • Comparar con el funcionamiento de la repetición simple, en tanto la repetición condicional evalúa una condición para determinar la cantidad de repeticiones. • Destacar la potencialidad -dentro de la creación de un juego- de poder programar una repetición desconociendo la cantidad de repeticiones que se producirán.
Contenido y desarrollo
Se retomará el programa utilizado en la clase 26 y se pedirá a las y los estudiantes que lo completen para que los jugadores ganen al acertar tres lanzamientos. Luego de unos minutos para que realicen sus programas, se discutirá entre toda la clase sobre las opciones exploradas. Se introducirá la repetición condicional y el uso de “While” o “Repetir mientras que” y se trabajará sobre casos en los que la cantidad de repeticiones no son conocidas de antemano, para remarcar su diferencia con la repetición simple vista anteriormente. Este punto se aprovechará para explorar el azar en la definición de un juego. Por otro lado, se retomará la idea de condición, vista con la alternativa condicional que -en este caso- será la que indique en qué momento detener la repetición de las instrucciones contenidas dentro del bloque. Para finalizar, se incluirá una nueva condición al desarrollo del juego, limitar el número de intentos, lo que implicará que las y los estudiantes incluyan el uso de una nueva variable. Ver Actividades.
Herramientas
Alice

Clase 28 - Proyecto integrador II

Objetivos
• Continuar con el trabajo en el proyecto final.
Contenido y desarrollo
En esta clase se continuará con el trabajo en el proyecto final, aprovechándose para resolver dudas y consultas, y monitorear el progreso. Ver Actividades.
Herramientas
Alice

Clase 29 - Proyecto integrador III

Objetivos
• Presentar proyecto final.
Contenido y desarrollo
En esta clase los distintos grupos mostrarán su proyecto final, presentándolo para el resto de la clase. Ver Actividades.
Herramientas
Alice

Secuencia de actividades

Clase 1 - Introducción

El/La docente hará su presentación y una breve descripción de la materia.

En caso de no tener Alice[3], se procederá a su instalación en todas las computadoras disponibles, mientras tanto, el/la docente puede realizar una pequeña encuesta para indagar sobre opiniones, relación con la tecnología, conocimientos previos y cualquier información que considere relevante, por ejemplo, para establecer canales de comunicación con las y los estudiantes.

Una vez concluida la instalación, se preguntará a las y los estudiantes qué cosas creen que son necesarias para hacer una película. Se espera que hablen de luces, cámaras, personajes, escenografía y especialmente, que se mencione el concepto de **guión**. En caso de que no lo propongan, puede orientarse a las y los estudiantes con preguntas como: ¿y cómo saben los actores lo que tienen que hacer? ¿La película cuenta una historia? ¿Dónde está esa historia antes de que empiece la grabación?

Luego de ese intercambio, se les contará que Alice es una herramienta que sirve para hacer sus propias animaciones y juegos, se les preguntará por animaciones y juegos que conozcan y cómo y por quién piensan que están hechas.

Se comenzará a mostrar el entorno, en este punto es muy importante detallar cómo cambiar el idioma si no se encuentra en español.

Se agregarán personajes, se indicará cómo se ven en el árbol de objetos y aparecen en el escenario.

Una vez agregados uno o dos personajes, se preguntará qué piensan que va a pasar en la animación y se apretará el botón “Ejecutar”. Se indicará el lugar donde va el guión. La conclusión a la que se debería arribar es que si el guión está vacío, nada ocurrirá al comenzar la ejecución.

En este momento las y los estudiantes podrán familiarizarse con la herramienta y explorar las galerías y características de la misma. Se pedirá a las y los estudiantes que hagan que dos personajes se muevan por el escenario.

Finalmente, se mostrará cómo hacer que un personaje haga algo, arrastrándolo desde el árbol de objetos al espacio del guión y eligiendo la acción deseada.

---

[3] Alice es una herramienta de programación para adolescentes que puede descargarse de http://www.alice.org/ de manera gratuita. Para facilitar su uso en las netbooks recomendamos descargar la versión 2.4. Se trata de la opción mencionada como "Current Release with complete Spanish Gallery- 2.4.3" en http://www.alice.org/index.php?page=downloads/download_alice2.4. La herramienta se descarga en inglés y luego mediante una opción se la pasa a nuestro idioma.

Clase 2 - Noción de programa y proyecto simple I

La clase comenzará con una propuesta simple en Alice: hacer una animación que incluya un personaje y un vehículo en la que el vehículo se aproxime a gran velocidad y cuando esté por chocar al personaje lo esquive y retome su camino. Además, se pedirá a las y los estudiantes que respondan algunas preguntas:

- ¿Qué ocurre si se intercambia el orden de las instrucciones?
- ¿Cuál es la diferencia entre arrastrar un método al espacio del guión y hacer clic derecho en un personaje del árbol de objetos, seleccionándolo desde ahí?
- ¿Para qué usarían cada caso?

Estas preguntas serán el pie para explicar que el guión es un programa, ver que las instrucciones se ejecutan en orden, de a una por vez y distinguir entre el momento en que se escribe y el momento en que se ejecuta nuestro programa.

Adicionalmente se presentará un proyecto grupal en el que las y los estudiantes deberán crear una animación o videojuego que aborde una temática (de su interés, a elección del docente o en el marco de algún proyecto que esté abordando la institución). El proyecto se hará fuera del horario de clase y se retomará más avanzado varias clases más adelante.

Durante el resto del trimestre se reservarán pequeños espacios de las clases para preguntar por el nivel de avance del proyecto, incentivar su continuación y resolver dudas. Para que el proyecto sirva como ejercitación de algunas de las temáticas de programación que se verán en la primera parte de la materia, será parte de la consigna que el proyecto incluya algún tipo de variabilidad que deberá poder determinarse en tiempo de ejecución. Por ejemplo, en base a una pregunta al usuario o botón a apretarse al comienzo, la historia podrá terminar bien o mal, transcurrir de día o de noche, etc.

Clase 3 - Métodos I

Se presentará el escenario “Encuentro de tercer tipo”[4] y se les propondrá a las y los estudiantes que creen un programa a partir de un guión disponible en el archivo “Guión del encuentro de tercer tipo”. (Ambos archivos deberán ser compartidos a las y los estudiantes a través de un pendrive o Internet).
Para comenzar la actividad se leerá en conjunto y voz alta la historia:

Después de viajar por el espacio, una nave tripulada por un robot acaba de aterrizar en la Luna. El robot instala en la superficie una cámara que nos permite ver la escena: el robot, la nave espacial y la Luna. Lo primero que se observa es al robot emitiendo un sonido de exploración. De repente un extraterrestre aparece de atrás de unas rocas, y dice "Bienvenidx". El robot encara al alien, pega un salto y gira su cabeza de la sorpresa, vuelve a emitir el mismo sonido, se acerca al extraterrestre para inspeccionarlo, luego mira hacia la cámara, vuelve a emitir el sonido de exploración, la cabeza del robot se pone roja y dice: "¡Houston, tenemos un problema!".

Se les propondrá a las y los estudiantes que comiencen a programar el guión.

A medida que trabajen con la historia, se observará que la secuencia de acciones de los personajes -una atrás de la otra- se vuelve extensa y complicada de leer.

Luego de que hayan avanzado con el programa, se interrumpe la clase, y se les dice a las y los estudiantes que el guión cambió y se les indica que ahora el robot primero inspecciona al alien y luego se sorprende.

Se deja que las y los estudiantes cambien su programa a partir de la nueva consigna.

Transcurrido un tiempo se detendrá la actividad y se volverá sobre la historia para trabajar en conjunto con las siguientes preguntas disparadoras:

- ¿Cómo podríamos haber escrito el programa para simplificar los cambios necesarios al modificar la historia?
- ¿Cómo podemos subdividir el comportamiento del robot en acciones más abarcativas?

Se espera que las y los estudiantes identifiquen acciones más generales que involucren varios movimientos del robot. Por ejemplo, es posible identificar que el robot -de acuerdo al guión inicial- se sorprende, investiga y reacciona. Otras respuestas también pueden ser correctas.

Se propondrá buscar -por ejemplo- la acción "investigar" dentro de las acciones que puede realizar el personaje (solapa métodos). Como no se encontrará la instrucción investigar, se volverá

[4] Actividad basada en “Learning to program with Alice” de Wanda Dann, Stephen Cooper y Randy Pausch.

a preguntar: ¿qué movimientos tiene que llevar a cabo el robot para “investigar” al alien recién descubierto de acuerdo al guión?

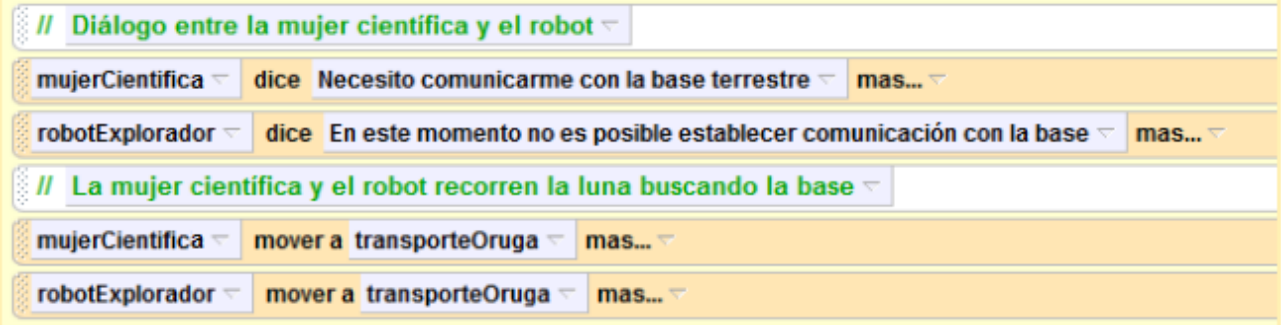
La propuesta será entonces modificar los programas para incluir estas acciones más generales bajo la forma de métodos. Es decir, programar a partir de la identificación de acciones de un mayor nivel de abstracción que involucren una serie de pasos más simples y cuenten con un fin específico.

Se les pedirá a las y los estudiantes que armen un método para cada acción identificada, con un nombre ilustrativo y que lo incluyan en sus programas. Y una vez definidos los métodos para que el robot cumpla con la totalidad del guión, se ejecutará la animación.

En muchos casos es probable que se programen las instrucciones dentro de los métodos pero sin invocarlos, por lo que al ejecutar la animación el robot se quedará quieto. En otros casos es posible que se invoquen los métodos nuevos pero no se incluya ninguna instrucción dentro de ellos, por lo que el robot también se quedará quieto.

Ante consultas de este tipo es importante indicar que en la estrategia de creación de un método son necesarias dos instancias. Por un lado especificar las instrucciones concretas que componen esa acción más abstracta. Y por otro llamar o invocar al método nuevo en el lugar adecuado dentro de nuestro programa.

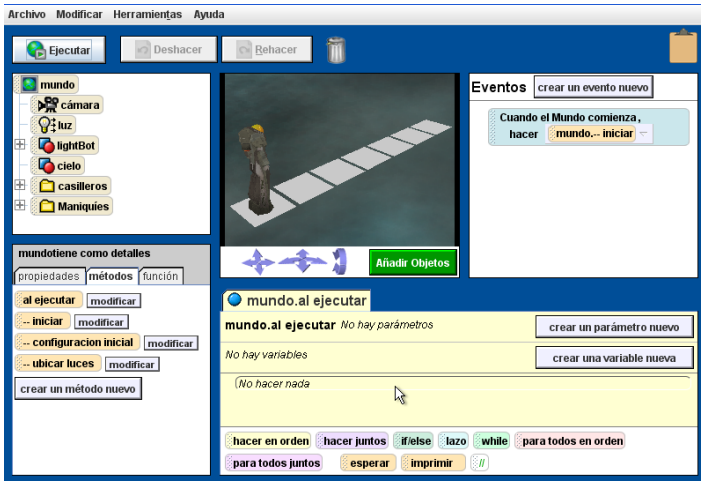
Para cerrar, vale la pena plantear el hecho de que los programas que realizamos no sólo se deben hacer entendibles para la computadora sino que también es importante que otra persona pueda leer nuestro programa y entenderlo. Para facilitar la lectura, organización y comprensión de los programas es recomendable utilizar **comentarios**, notas escritas para describir brevemente el comportamiento de las instrucciones que componen una parte del código pero que durante la ejecución del programa son ignoradas. Están destinados a orientar a quienes leen el programa, no son indicaciones para la computadora. Los comentarios en Alice comienzan con **//** y se ven en color verde. Para incluirlos se debe arrastrar el bloque // desde la barra inferior hasta el lugar del guión que se prefiera y escribir el comentario dentro de él. A continuación se ve un fragmento de código que incluye comentarios.



Clase 4 - Alternativa condicional

Se repasará el concepto de método utilizando el escenario inicial disponible en Lightbot.a2w[5], que las y los estudiantes deberán abrir en Alice.

Luego de cargar el archivo se obtiene el siguiente escenario inicial:



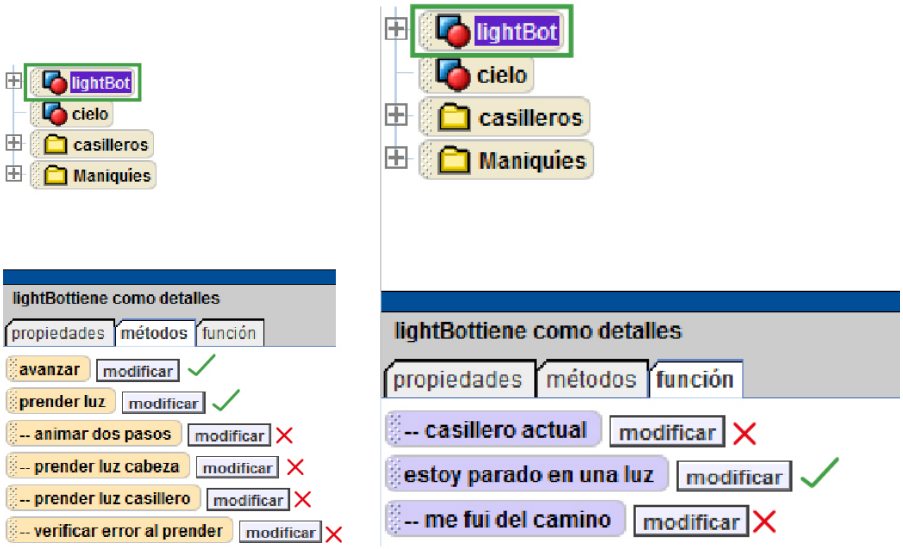
Escenario inicial de Alice con la actividad LightBot.a2w

Una vez hecho esto, se les pedirá a las y los estudiantes que ejecuten reiteradas veces el programa, y que tomen nota de los cambios que se producen en cada ejecución. A partir de este escenario inicial las y los estudiantes deberán programar al robot Lightbot para que avance de a un casillero por vez. En cada casillero Lightbot deberá ver si la luz está prendida o apagada, y en el caso de que esté apagada prenderla para poder continuar.

Antes de comenzar, se repasará en conjunto con las y los estudiantes el concepto de método visto la clase anterior y se señalarán los métodos disponibles (“avanzar”, “prender luz” y “estoy parado en una luz”) para cumplir la consigna. Para ello se deberá tener en cuenta que únicamente deberán usarse aquellos métodos que no comienzan con “--”. Estos últimos son parte de la configuración inicial, por lo que no será necesario utilizarlos en la resolución del problema.

A continuación se muestra los métodos que pueden usarse:

[5] Esta actividad está inspirada en los juegos educativos de Lightbot. Más información en https://lightbot.com/



Los métodos tildados con verde son los que pueden usarse para resolver el problema. Los que empiezan con un doble guión "--" son parte de la configuración inicial.

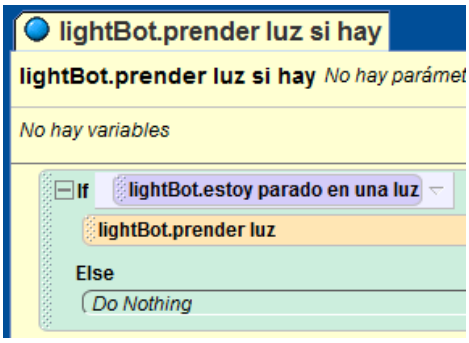
Se insistirá en el funcionamiento cambiante del escenario en cada ejecución y se les pedirá a las y los estudiantes que hagan un programa que resuelva la consigna.

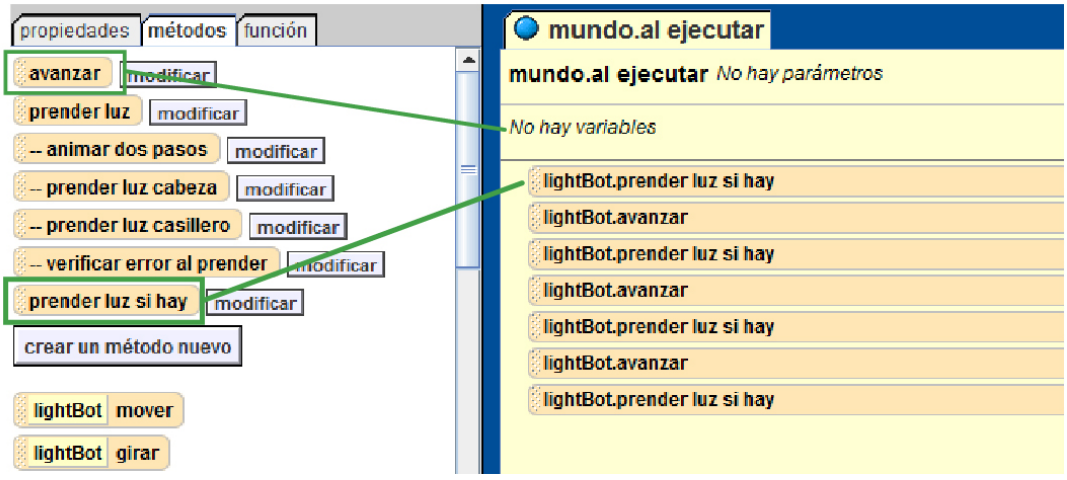
Luego que realicen varios intentos para resolver el ejercicio, es probable que las y los estudiantes encuentren dificultades a la hora de definir cuándo encender o no la luz, dado que las luces a encender cambian en cada ejecución. Se presentarán las alternativas condicionales como la forma en que la computadora evalúa la información que tiene en ese momento de la ejecución y toma un camino en consecuencia. En este caso, nos permitirá evaluar si la luz de un casillero se encuentra encendida o apagada.

Se pedirá a las y los estudiantes ejemplos de la vida cotidiana (Si llueve llevo paraguas, Si tengo hambre me como un alfajor, etc.) y ejemplos en juegos o aplicaciones que conozcan (Si Mario toma el hongo cambia de tamaño, si me disparan pierdo puntaje, si pongo la contraseña incorrecta no puedo entrar a Facebook, etc.)

Se presentará la instrucción if/else de Alice explicando la traducción "Si… entonces…" y se propondrá terminar el ejercicio. A partir de la instrucción if/else es posible evaluar -para cada casillero- si el personaje está parado en una luz, y sólo en ese caso encenderla y avanzar. Se insistirá a las y los estudiantes acerca de la pertinencia de usar un método nuevo para llevar a cabo esa evaluación, que podrá llamarse "prender luz si hay".

Una solución posible para el problema es la siguiente:

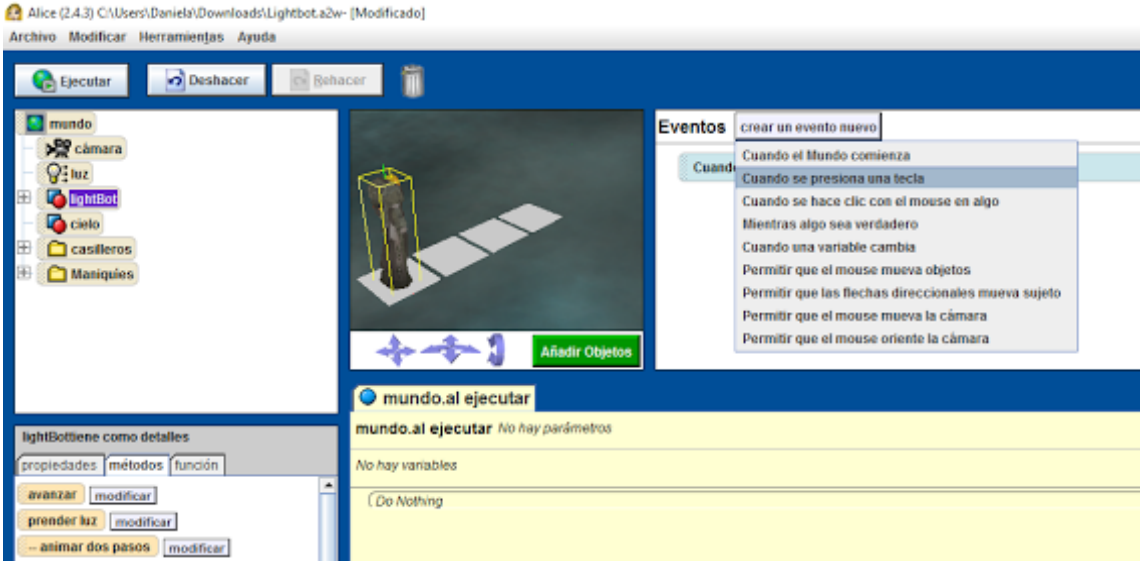




Para finalizar, se mostrarán brevemente los eventos del teclado para que puedan explorar opciones de interactividad y puedan programar juegos interactivos.
Se preguntará a las y los estudiantes cuál creen que es la diferencia entre un juego y una animación. Se espera que la noción de interactividad surja de la clase, y se resaltará que hasta este punto se han construido animaciones con las que el usuario no interactúa. Es decir, al ejecutar los programas se observan con exactitud la sucesión de instrucciones que se han programado. Los eventos dan la posibilidad de que el usuario interactúe con el programa durante la ejecución y que las acciones de los personajes respondan a, por ejemplo, clics del mouse o teclas presionadas por el usuario.

Para ilustrar este punto se les mostrará a las y los estudiantes una modificación al programa anterior con el uso de eventos del teclado. Para convertir la animación en un juego, se hará que cuando el usuario presione la flecha derecha del teclado Lightbot avance un casillero. Es decir, la presión de una tecla como evento provocará como respuesta la ejecución del método avanzar.

El/la docente mostrará que para introducir eventos en Alice basta con hacer clic en “Crear un evento nuevo”, seleccionar qué tipo de acción lo provoca (cuando el usuario toca las teclas, hace clic o mueve el mouse, por ejemplo) y qué se espera que realice el programa.



Para concluir con esta actividad se pedirá a las y los estudiantes que agreguen un segundo evento: cuando el usuario presione la barra espaciadora el personaje se deberá fijar si hay o no una luz en ese casillero.

Se avisará a las y los estudiantes que deberán traer el avance del proyecto presentado en la clase 3 para la clase siguiente.

Clase 5 - Proyecto simple II

En esta clase se trabajará sobre el proyecto presentado en la clase 2. Cada grupo expondrá su idea original, el nivel de avance y si encontraron nuevas herramientas o alguna limitación al momento de programar su pieza. Se propondrán ideas y objetivos a futuro y se realizará un intercambio para abordar de manera conjunta los aspectos de la herramienta que requiera cada pieza.

Clase 6 - Variables I

Se preguntará la diferencia entre juegos y animaciones. El objetivo es que las y los estudiantes se aproximen a la idea de interactividad. Se retomará la diferencia entre tiempo de escritura y tiempo de ejecución visto en la clase 2 y se les mostrará cómo hacer que el usuario ingrese su nombre durante el transcurso del programa y algún personaje lo diga.

Como ejercicio para introducir variables, se pedirá que en un mundo con 3 personajes hagan que ingresando una única vez el nombre del usuario todos los personajes lo saluden.

Las y los estudiantes se encontrarán con la dificultad de no poder recuperar el nombre ya que no fue guardado en ningún lado. Esto nos dará una buena oportunidad para presentar el concepto de variable y hacer una primera aproximación a la idea de memoria y almacenamiento.

Para ilustrar la dificultad que encontraron para recuperar el nombre, puede realizarse una representación con la colaboración de 3 estudiantes:

1. Se le dará un papel al primer estudiante para que anote un nombre.
2. Se pedirá al segundo estudiante que lo lea.
3. Se tirará el papel.
4. Se pedirá al tercer estudiante que también lea el nombre, pero ya no tendrá de donde hacerlo.

Luego de repetir esta situación dos veces, se hará lo mismo pero incorporando una caja o sobre que será una representación de la variable:

1. Se le dará un papel al primer estudiante para que anote un nombre.
2. Se guardará el papel en la caja/sobre.
3. Se pedirá al segundo estudiante que lo lea.
4. Se guardará el papel en la caja/sobre.
5. Se pedirá al tercer estudiante que lea el nombre.

Se discutirá con el curso cómo creen que debería llamarse la caja/sobre: ¿con el nombre que anotó el primer estudiante? ¿Qué pasa si se repite la secuencia pero preguntando el nombre al segundo estudiante?

Se recreará otra situación en la que se me modifica el nombre que se guarda en la caja:

1. Se le dará un papel al primer estudiante para que anote un nombre.
2. Se guardará el papel en la caja/sobre.
3. Se pedirá al segundo estudiante que lo lea.
4. Se guardará el papel en la caja/sobre.
5. Se pedirá al tercer estudiante que anote un nombre.
6. Se guardará el papel en la caja/sobre en lugar del otro
7. Se pedirá al primer estudiante que lo lea.

Con esta dramatización deberá quedar en claro que denominar a la caja con alguno de los nombres en particular no tiene sentido y se escribirá “Nombre” en ella. Luego se explicará que la caja representa una variable, que es un espacio de memoria donde se almacena información para poder recuperarla o modificarla y la importancia de que se la nombre con claridad (en función del tipo de contenido que tendrá y no con un contenido en particular).

Se mostrará como crear y utilizar variables en Alice y se invitará a las y los estudiantes a terminar el ejercicio propuesto.

Clase 7 - Diferencia entre programa y algoritmo

Se comenzará recapitulando brevemente los conceptos de programación que se vieron hasta la clase pasada: métodos, alternativa condicional y variables. Este repaso resulta conveniente puesto que todas las actividades requerirán utilizar estos conceptos para su resolución.

A continuación se propondrá realizar en parejas el siguiente ejercicio en Alice:

- Un personaje pregunta al usuario por el resultado de un juego entre dos equipos en el que pueda haber un ganador o empate (fútbol, básquet, handball, rugby, etc.). El usuario deberá primero ingresar el puntaje del primer equipo y luego le puntaje del segundo. El personaje de Alice debe decidir si ganó el primer equipo, el segundo o hubo empate.

Para poder resolver este ejercicio será necesario utilizar interactividad (“preguntar al usuario un número”), variables (para guardar ambos puntajes) y, al menos, dos alternativas condicionales (if/else) para determinar si hubo un ganador o hubo empate. Notar que solamente con una única alternativa condicional no sería posible discernir entre 3 resultados posibles.

A medida que el/la docente vaya recorriendo las mesas de trabajo y notando que todos los grupos pudieron avanzar en la actividad, se propondrá una puesta en común en donde se analicen distintas propuestas de solución (dos if/else anidados, 3 if/else secuenciales, etc.) y si la siguiente es una solución válida:

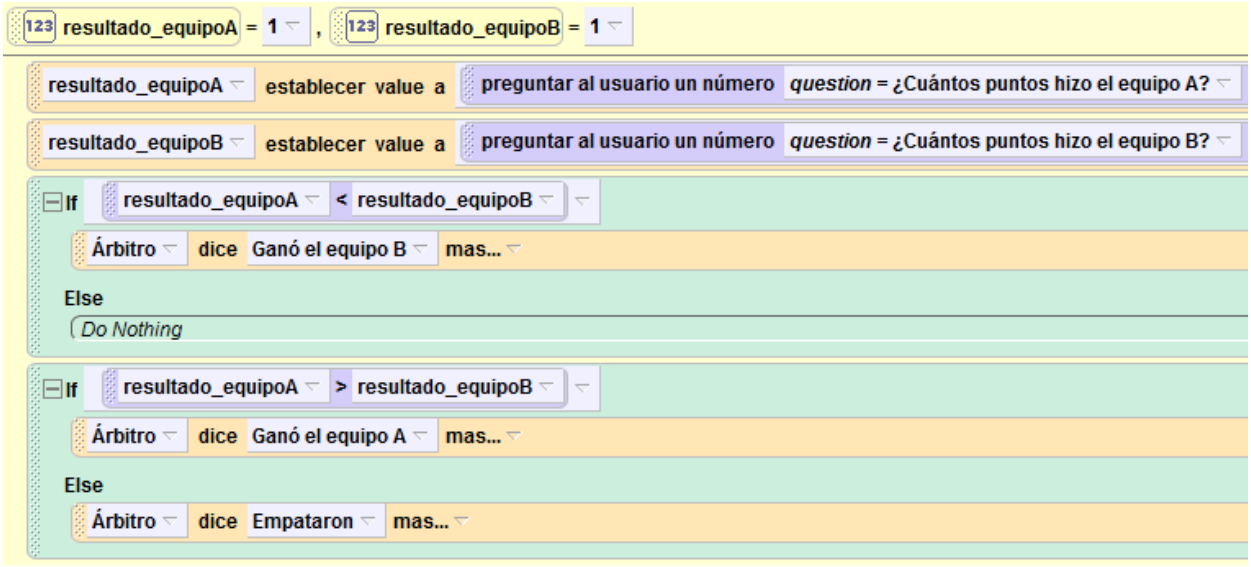


Figura 7.1

Algunas preguntas que pueden orientar la discusión del programa de la figura 7.1:

- ¿Podemos saber si el programa funciona bien sin ejecutarlo en Alice?
- ¿Qué posibles resultados deberíamos considerar para realizar un análisis?
- Si en el segundo condicional reemplazáramos el “>” por un “≥”, ¿qué pasaría?

- Si el programa tuviera muchas más líneas y variables, ¿creen que sería fácil detectar un problema de este tipo?

Aquí se puede hacer una breve digresión acerca de que a este tipo de defectos se los conoce como bugs (*bichos* en inglés)[6] y que al programar, como en cualquier actividad humana, suelen cometerse errores que generan defectos que hacen que nuestro programa no se comporte de la manera deseada.

Una buena práctica, que resulta de utilidad antes de comenzar a programar, es diseñar un algoritmo que resuelva el problema. ¿Qué es un algoritmo? Hay muchas definiciones distintas pero simplemente diremos que **un algoritmo describe una estrategia de resolución de un problema reconociendo cuáles son sus partes centrales**. Para definir la estrategia se puede (y sugerimos) utilizar el castellano con oraciones que describan un comportamiento específico.

Se puede pensar entre todos cuál sería un algoritmo para el ejercicio anterior. Daremos el siguiente a modo de ejemplo:

Preguntar al usuario los resultados de A y B
Si A es más grande que B
    Ganó A
Si A es más chico que B
    Ganó B
Si A y B son iguales
    Empataron

A este lenguaje a medio camino entre el castellano que usamos cotidianamente y un programa en un lenguaje de programación lo llamaremos *pseudocódigo*. Diseñar una estrategia de resolución o algoritmo, nos permite pensar de manera más abstracta cuáles serían las grandes acciones que deberíamos realizar para resolver el problema. Con el algoritmo diseñado resulta más sencillo pasar luego a un programa.

A continuación se les propondrá a las y los estudiantes que diseñen un algoritmo para resolver el siguiente problema:

- Se dispondrán 4 personajes en el escenario. Uno de ellos les preguntará la edad a los otros 3 y determinará quién es el más grande. Las edades las deberá ingresar el usuario a medida que se vayan preguntando, es decir, no están fijadas en el código del programa. Notar que debe decir el nombre del personaje de mayor edad y no cuál de las edades es la más grande.

El/La docente deberá prestar atención y enfatizar que la idea no es programar la solución en Alice ni escribirla en papel con las mismas instrucciones que Alice provee. A medida que vayan realizando los algoritmos, el/la docente puede motivar a las y los estudiantes a que chequeen si su

[6] https://es.wikipedia.org/wiki/Error_de_software#Or.C3.ADgenes_del_t.C3.A9rmino

algoritmo funcionaría para distintos casos: que el primero sea el más grande, el segundo o el tercero. Si hubiera dos personajes con la misma edad, ¿qué haría su algoritmo?

Luego de que lo hayan terminado, se realizará una puesta en común y se analizarán las distintas propuestas de algoritmos:

- ¿Todos devuelven un nombre?
- ¿Todos funcionan?
- En el caso de que hubiera edades repetidas, ¿todos devuelven el mismo nombre?

Es interesante ver que para el mismo problema hay distintos abordajes o estrategias de resolución, notando que, por ende, también habrá muchos programas que lo resuelvan (si los algoritmos propuestos por las y los estudiantes fueran similares, el/la docente puede proponer uno diferente y sumarlo al análisis).

Como cierre, el/la docente comentará que los algoritmos no se restringen sólo a la programación sino que en nuestra vida cotidiana también los usamos y diseñamos:

- Cuando dividimos dos números **usamos** el algoritmo de la división.
- Cuando damos las indicaciones para llegar de un punto a otro **diseñamos** un algoritmo que indica cuáles son los pasos que debemos realizar para llegar a destino.
- Si para cocinar **usamos** una receta de cocina, estamos siguiendo los pasos definidos en ella para obtener el plato que queremos, es decir, un algoritmo culinario.

Por lo tanto, es importante diferenciar entre algoritmo y programa: mientras el primero determina una serie de pasos para resolver un problema de manera automática y utiliza el lenguaje castellano, el segundo también determina una serie de pasos o instrucciones pero éstas se ejecutan en una computadora y se definen en un lenguaje de programación como Alice, es decir, un lenguaje que la máquina conoce.

Clase 8 - Lenguajes de programación

Durante esta clase se usarán herramientas que ya vienen instaladas en Huayra y no vienen por defecto en Windows. Por este motivo, para no dedicar tiempo a la instalación de estas herramientas, sugerimos usar durante toda la clase el sistema operativo Huayra. Cada estudiante deberá tener una carpeta en el directorio "alumno" o en el "Escritorio"[7] con los siguientes archivos: `compilar_y_ejecutar.sh`, `funciones.h`, `funciones.c` y `entrada_de_cine.c`. Los archivos se encuentran disponibles en el siguiente link: https://goo.gl/hqZRj0. Para descargar todos los archivos, al abrir el link, arriba a la derecha aparecerá una flecha similar a . Al hacer clic allí, se descargará el archivo "`TI3_clase_8_archivos.zip`", que contiene comprimidos los 4 archivos mecionados. Para extraerlos, cliquear en el archivo con el botón derecho del mouse y luego en "Extraer aquí".

Para poder ejecutar el archivo "`compilar_y_ejecutar.sh`" haciendo doble clic en él es necesario agregarle permisos de ejecución[8]. Para ello, en cada compu, hay que hacer clic derecho en el archivo "`compilar_y_ejecutar.sh`", ir a "Propiedades", luego "Permisos"[9] y, por último, hacer clic en "Permitir ejecutar el archivo como un programa". Luego de hacer doble clic en el archivo se deberá elegir la opción "Ejecutar".

Al comenzar, se recordará brevemente lo que hicieron la clase pasada: dado un problema, aprendieron a diseñar un algoritmo en un lenguaje denominado pseudocódigo.

Como primera actividad, se pensará entre todos un algoritmo que resuelva el siguiente problema:

- Dory comenzó hace pocos días a trabajar como vendedora de entradas en el viejo cine "El Cambalache submarino". Como tiene especialmente poca memoria, quiere hacer un programa que le diga, dada la edad de una persona, cuál es el valor de su entrada. Para los que tienen 5 años o menos y para los que tengan 65 años o más, la entrada cuesta 20 caracolas. Para el resto, la entrada tiene un valor de 50 caracolas.

Un algoritmo posible es:

Preguntarle la edad al usuario
Si tiene 5 años o menos
    Paga 20
Si tiene 65 años o más
    Paga 20
En cualquier otro caso
    Paga 50

[7] No guardar en la carpeta "Documentos" ya que no se pueden cambiar los permisos del script "`compilar_y_ejecutar.sh`".
[8] Wiki sobre permisos de archivos en Huayra: http://wiki.huayra.conectarigualdad.gob.ar/index.php/Permisos
[9] En las clases 17 y 18, en donde se trabajará sobre sistemas operativos, se dará una noción de qué son los permisos y para qué sirven.

Una vez generado consenso acerca de cuál es el algoritmo que resuelve el problema de Dory, se propondrá programarlo. Pero esta vez no será en Alice sino en otro lenguaje de programación llamado C. A diferencia de Alice, en C las instrucciones del programa hay que escribirlas en modo texto en un archivo que se conoce como "fuente".

En esta clase (y en este curso) se utilizará una parte mínima del lenguaje C, de manera muy simplificada[10]. Para introducir a las y los estudiantes en la sintaxis del lenguaje y cómo ejecutar los programas, este primer problema lo programarán entre todos junto con el/la docente. Una buena manera para empezar es observar qué es lo primero que hay que hacer según el algoritmo que escribieron recién. Si lo tuvieran que programar en Alice, ¿qué harían primero? Probablemente haya consenso en que habría que "preguntar al usuario un número" para conocer cuál es su edad. En la versión simplificada del lenguaje que usaremos, pedirle al usuario que "ingrese su edad", en C lo podemos hacer del siguiente modo:

```
imprimir_un_texto("Ingrese su edad:");
int edad = leer_entero();
```

`imprimir_un_texto()` es una instrucción que imprime por pantalla solamente texto, el cual se escribe entre comillas dobles.

En C, al igual que en Alice, cuando queremos crear y usar una variable, debemos indicar si se trata de un número, de un texto, o de algún otro tipo de valores. En Alice, cuando creábamos la variable se nos ofrecía una lista con los posibles tipos[11] que podía tener. En C, para crear una variable indicamos primero su tipo -**`int`**- y luego su nombre, **`edad`** en este caso. Además, al crearla, podemos asignarle un valor. Por ejemplo, la edad que ingresa el usuario mediante la instrucción **`leer_entero()`**. Si lo quisiéramos hacer en dos pasos, primero crear la variable y luego asignarle un valor, el código quedaría del siguiente modo:

```
imprimir_un_texto("Ingrese su edad:");
int edad;
edad = leer_entero();
```

Cabe destacar que en C todas las instrucciones terminan con un punto y coma. ¿En Alice cómo se separaban las instrucciones? Otro aspecto que se observa en estas pocas instrucciones es que la función que imprime un texto tiene entre paréntesis las palabras que van a ser impresas pero, en cambio, la que lee la edad que ingresa el usuario no tiene nada entre paréntesis. Esto se debe a que la instrucción que imprime necesita conocer el valor que va a escribir en la pantalla pero la instrucción que lee la edad del usuario no necesita ningún valor previo para poder ejecutarse, simplemente se fija qué valor ingreso el usuario y lo devuelve.

Al finalizar esta construcción colectiva de las primeras líneas de código, se les puede pedir a las y los estudiantes que escriban estas instrucciones en su archivo "`entrada_de_cine.c`". Las

[10] Para no abordar en esta instancia algunos aspectos engorrosos de la sintaxis de C, como puede ser leer un valor que ingresa el usuario o imprimir un texto por pantalla, en el archivo "`funciones.c`" ya están programadas algunas funciones que nos van a permitir realizar estas operaciones de manera más sencilla.
[11] Recordemos que el *tipo* de una variable indica qué valores puede almacenar y qué operaciones se pueden hacer entre ellos. Por ejemplo, las variables enteras o *int* almacenan números mientras que las de tipo texto o *string* almacenan texto diverso.

mismas deben ir debajo de la línea que dice "`// ESCRIBIR EL CÓDIGO AQUÍ`". El archivo se puede abrir en cualquier editor de texto que tengan instalado.

En este punto, se puede preguntar a la clase qué creen que va a ocurrir si ejecutan este programa dado que no tenemos personajes ni escenario. Para correr el programa, hay que hacer doble clic en el archivo "`compilar_y_ejecutar.sh`". Este archivo es un programa muy simple que recibe el archivo con el programa que hicieron, lo traduce a código de máquina para que lo podamos ejecutar[12] y lo ejecuta.

Lo que verán al ejecutar el programa será una pantalla negra, llamada *terminal*, en donde aparecerá el texto "Ingrese su edad:". El programa habrá ejecutado la primera de las instrucciones de nuestro programa y se encuentra esperando que el usuario ingrese un valor para la edad. Similar a lo que ocurría en Alice cuando aparecía un pop-up en donde ingresábamos un valor. Luego de ingresar un número y presionar "Enter", el programa terminará y no hará más nada, debiendo cerrar la ventana de la terminal a mano.

Para verificar que el número que ingresamos haya sido guardado correctamente en la variable, podemos agregar la instrucción **`imprimir_un_numero(edad)`**, obteniendo el siguiente programa:

```
imprimir_un_texto("Ingrese su edad:");
int edad = leer_entero();
imprimir_un_numero(edad);
```

A continuación se propone seguir construyendo entre todos el programa en C que resuelve el problema según el algoritmo que plantearon hasta llegar a tener un programa similar a:

```
imprimir_un_texto("Ingrese su edad:");
int edad = leer_entero();

int valor_entrada;
if (edad < 6) {
      valor_entrada = 20;
} else {
      ...
}
...
```

En los 3 puntos suspensivos las y los estudiantes deberían terminar de escribir el programa, para lo cual ya tienen todas las herramientas de sintaxis de C necesarias.

[12] Se profundizará este aspecto en la siguiente clase.

Un posible programa al que podrían arribar y que resuelve el problema es

```
imprimir_un_texto("Ingrese su edad:");
int edad = leer_entero();

int valor_entrada;
if (edad < 6) {
      valor_entrada = 20;
} else {
      if (edad > 64) {
            valor_entrada = 20;
      } else {
            valor_entrada = 50;
      }
}

imprimir_un_texto("El valor de su entrada es");
imprimir_un_numero(valor_entrada);
```

Se puede establecer la similitud entre la instrucción **`if/else`** que usaban en Alice y la instrucción **`if {} else {}`**, las cuales son equivalentes. Simplemente cambia la forma en que se escriben. Mientras en Alice para el **`if`** y para el **`else`** tenemos un lugar especial donde encastrar las instrucciones que queremos ejecutar en cada caso, en C ese espacio lo delimitamos con las llaves. Notar que el segundo **`if {} else {}`** está dentro del primer **`else {}`**. Esto es así puesto que si la persona no tuviera menos de 6 años -la condición del primer **`if {}`**- todavía no podemos saber si tiene más o menos de 65 años. Por ende, en el caso de que tuviera 6 años o más y entrásemos en el primer **`else {}`**, deberíamos preguntar si tiene 60 años o más para poder terminar de decidir si tiene que abonar una entrada de 20 caracolas o de 50.

Para terminar, el/la docente contará que además de Alice y C, existen muchos otros lenguajes de programación, y la elección de uno u otro depende de varios motivos: el contexto del problema, el lenguaje de moda en ese momento, el gusto del programador, etc. Sin embargo, con cualquier lenguaje de programación, ya sea visual como Alice o textual como C, podemos programar cualquier algoritmo.

Clase 9 - El lenguaje de máquina

Al igual que en la clase pasada, aquí también se sugiere utilizar el sistema operativo Huayra. Los primeros minutos serán dedicados a distribuir entre las y los estudiantes el archivo "`BARF.zip`" (disponible en https://goo.gl/GnIxyS) e instalar el programa BARF. Para ello, primero se debe descomprimir el contenido de "`BARF.zip`", luego hay que otorgarle permiso de ejecución al archivo "`instalarBARF.sh`" y, por último, al hacer doble clic en dicho archivo elegir la opción "Ejecutar", lo cual realizará la instalación del software. Al comienzo de la instalación se nos pedirá la clave del usuario Administrador, que será la contraseña por defecto de Huayra ("*alumno*") o la nueva que se haya definido si fue cambiada.

Una vez finalizada la instalación de BARF, el/la docente comenzará realizando una pequeña demostración. Abrirá la carpeta "`Entrada de cine`" en donde están los mismos archivos de la clase pasada y una solución posible al problema de la entrada de cine. Puede mostrar a la clase el código de la solución abriendo el archivo "`entrada_de_cine.c`". A continuación, se procederá a tomar nota de cuáles son los archivos que se hallan en dicha carpeta: "`entrada_de_cine.c`", "`funciones.h`", "`funciones.c`" y "`compilar_y_ejecutar.sh`". Luego, hacer doble clic en "`compilar_y_ejecutar.sh`" y ver qué ocurre en esa misma carpeta. Lo que pasará es que se creará el archivo "`entrada_de_cine`". ¿Por qué ocurre esto? ¿Qué hace ese nuevo archivo?

Para intentar dilucidar entre todos qué podría hacer este archivo, abrirlo con un editor de textos, como *gVim* o *LibreOffice Writer*[13], y ver si se puede inferir algo de su contenido. Como se trata de un archivo binario, todo lo que verán es una secuencia de caracteres que aparentemente no tiene sentido. ¿Qué significa todo este texto incomprensible? ¿Qué pasaría si agregamos una línea al final del código y volvemos a hacer doble clic en "`compilar_y_ejecutar.sh`"? ¿Cambia este archivo? Para que puedan comparar si hubo cambios en ese archivo antes y después de agregar una línea de código, se sugiere que previo de volver a compilar y ejecutar, el/la docente cambie el nombre del archivo por "`entrada_de_cine_anterior`" o alguna variante similar que dé cuenta que se trataba de la primera versión del archivo. Luego de realizar algún pequeño cambio en el código del programa (por ejemplo, agregar una instrucción que imprima un texto al final), compilar y ejecutar, se generará un nuevo archivo "`entrada_de_cine`". Si queremos comparar caracter a caracter este archivo con el anterior, va a resultar difícil y tedioso. La manera más sencilla es comparar los tamaños de los archivos. Si uno fuera más grande que otro, entonces no podrían contener el mismo texto. De hecho, el más grande será "`entrada_de_cine`". ¿Por qué ocurre esto? ¿Cuál será la relación entre el código del programa que hicieron y este archivo? Aparentemente si tenemos más líneas de código parecería que el archivo generado es más grande.

El/La docente contará a la clase que, efectivamente, existe una relación directa entre el código que escribieron en C y el archivo "`entrada_de_cine`". Este último es el mismo programa que ellas y ellos hicieron en C pero traducido al **lenguaje de máquina**. ¿Y qué es el lenguaje de máquina? Son un conjunto de instrucciones muy sencillas que la computadora sabe ejecutar. Pero este conjunto de instrucciones no es el mismo para toda computadora ni todo dispositivo. Por ejemplo, el lenguaje de máquina que usan sus computadoras de escritorio es diferente del

[13] Puede que otros editores de texto no permitan abrir el binario por cuestiones de formato.

lenguaje de máquina que usan los celulares. Esto se debe a diversas decisiones de diseño que toman quienes desarrollan cada dispositivo. Como programar en lenguaje de máquina suele requerir más tiempo ya que las instrucciones son muy simples y, por ende, el programa final requiere mayor cantidad de instrucciones, se diseñaron otros lenguajes de programación como C y como Alice que nos brindan instrucciones más ricas, es decir, que equivalen a realizar varias de las instrucciones de máquina en una sola instrucción de C o de Alice. Sin embargo, las computadoras no entienden ni C ni Alice sino que sólo pueden ejecutar programas escritos en su propio lenguaje de máquina. ¿Cómo se resuelve este problema de programar en un lenguaje que nos resulta más cómodo a nosotros pero que la máquina no sabe interpretar? Con un programa conocido como compilador que traduce entre un lenguaje (como C o Alice) y el lenguaje de la máquina en la que queremos ejecutarlo[14].

Recapitulando la microactividad que realizamos antes, teníamos un programa escrito en C en el archivo "`entrada_de_cine.c`". Luego hicimos doble clic en el archivo "`compilar_y_ejecutar.sh`" lo cual creó el archivo "`entrada_de_cine`" y ejecutó el programa. ¿Qué fue lo que pasó entonces? Al hacer doble clic en "`compilar_y_ejecutar.sh`" se ejecutaron 2 comandos:

1. Se compiló, es decir, se tradujo el programa escrito en C al lenguaje de máquina generando el archivo "`entrada_de_cine`". Este archivo contiene el programa que hicimos en lenguaje de máquina.
2. Se ejecutó dicho archivo lo que equivale a ejecutar el programa contenido en él.

Como se mencionó en la clase pasada, al código escrito en C o en un lenguaje que no sea de máquina se lo conoce como **código fuente** mientras que al código compilado se lo conoce como **código binario, código objeto** o **ejecutable**[15].

Al finalizar esta explicación, comenzaremos a trabajar con BARF, un software desarrollado por programadores argentinos, de código abierto[16] y gratuito que permite, entre muchas otras cosas, visualizar (técnicamente *desensamblar*) el código de máquina producido luego de compilar un programa hecho en un lenguaje como C. Para ejecutar BARF basta con hacer doble clic en el archivo "`ejecutarBARF.sh`"[17]. Lo que hará BARF es leer el archivo "`entrada_de_cine`" y generar distintos gráficos en una carpeta con nombre "`entrada_de_cine_cfg`".

La primera actividad se realizará entre todos y consistirá en abrir y analizar el gráfico "`main.pdf`", ubicado en la carpeta mencionada en el párrafo anterior, el cual muestra nuestro programa en el lenguaje de la máquina que estamos usando. En la figura 9.1 se observa un gráfico similar o igual al que se verá en la clase. ¿Qué reconocen o pueden inferir de él?

---

[14] Existen otras soluciones que determinan otro tipo de lenguajes conocidos como *interpretados* sobre los cuales no hablaremos en este curso.
[15] En realidad binario, objeto y ejecutable no significan lo mismo aunque en esta instancia no se realizarán las distinciones. El código objeto es el resultado de compilar las partes de un programa por separado. Al juntar todos los códigos objeto se obtiene un ejecutable. El compilador mismo realiza estos 2 procesos. Tanto un objeto como un ejecutable son archivos binarios porque su representación no es textual sino en ceros y unos.
[16] Se profundizará sobre el concepto de software libre en la clase 19.
[17] Si no tuviera permisos de ejecución, habría que realizar el mismo procedimiento que con el archivo "`instalarBARF.sh`".

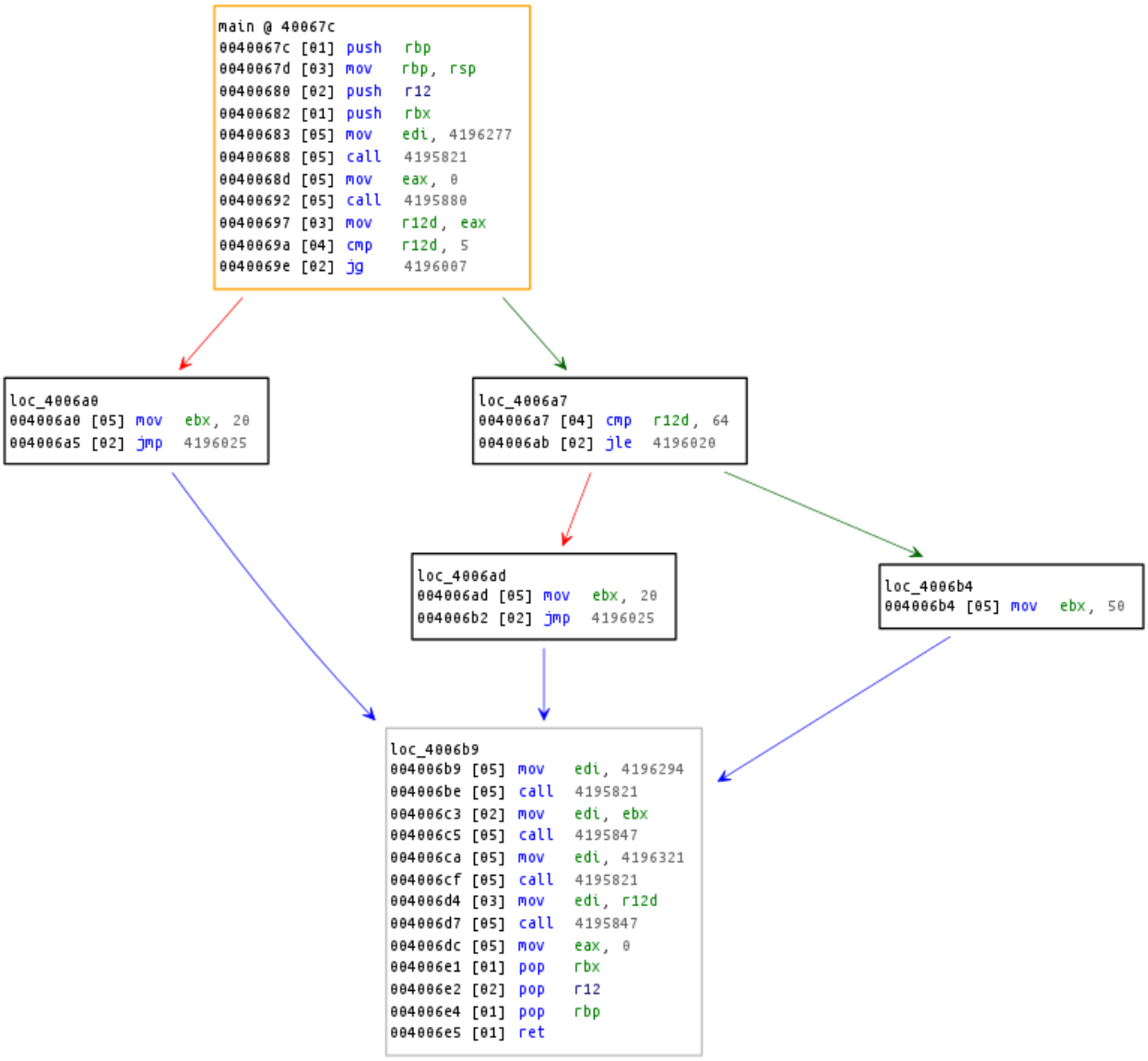


Figura 9.1

A continuación se detallarán algunas características que las y los estudiantes pueden notar o acerca de las cuales se les puede preguntar:

- En el gráfico aparecen los números 5, 20, 50 y 64, los mismos que en el código fuente.
- Esos números aparecen en las mismas líneas que códigos en azul que dicen “mov” y “cmp”.
- Los códigos en azul son en realidad instrucciones del lenguaje de máquina y, generalmente, abrevian una palabra más larga y en inglés. ¿Con qué palabra del español podemos relacionar **mov**, por ejemplo? ¿Y **cmp**? Es probable que **mov** lo relacionen con palabras derivadas del verbo mover mientras que **cmp** puede resultar en una mayor cantidad de palabras posibles.
- Al igual que hacían en Alice o en C para guardar un valor en una variable, en lenguaje de máquina la instrucción que nos permite guardar una valor en la memoria es **mov**, es decir, “mueve/copia/guarda” un valor a un lugar de la memoria.

- El **`cmp`**, en cambio, viene de “comparar” dos valores. En las alternativas condicionales de su código C, es decir, en las condiciones de los **`if {} else {}`** ellas y ellos compararon por mayor o por menor (o por mayor e igual y menor e igual) la edad con otro valor (5 o 6 y 64 o 65).
- Al igual que el **`mov`** y el **`cmp`**, el resto de las instrucciones del lenguaje de máquina realizan tareas muy simples.
- Un elemento interesante para trabajar son las flechas que relacionan distintos bloques de código. Éstas expresan los 2 caminos posibles de una instrucción **`if {} else {}`**. El primer caso corresponde al primer **`if {}`** en el que se chequea si la edad es mayor a 6. Si es menor a 6 años, entonces guardo el valor 20 en la variable `valor_entrada` y va directamente a la penúltima línea. Esto se indica mediante la flecha roja, es decir, el rojo nos dice que se entró al **`if {}`** y el verde que se entró al **`else {}`**. Si seguimos por la primera flecha verde, es decir, si la edad es mayor o igual a 6 años, se vuelve a ejecutar otro **`if {} else {}`** y por eso vuelven a aparecer otras dos flechas roja y verde. La flecha azul indica a qué instrucción debemos ir una vez que terminamos de ejecutar un **`if {}`** o un **`else {}`**. En todos los casos de este programa, siempre se va a la penúltima instrucción.

El resto de los valores y las instrucciones que aparecen no se trabajarán durante el curso puesto que el objetivo no es aprender lenguaje de máquina sino conocer cómo hace la computadora para ejecutar un programa. Simplemente se puede agregar que el resto de las palabras azules son otras instrucciones muy simples como el **`mov`** y el **`cmp`**, y que el resto de los valores son indicaciones que ayudan al programador que necesita leer el código de máquina a entender en profundidad qué es lo que hace el programa.

La segunda actividad consistirá en sugerirle a las y los estudiantes que realicen algún cambio pequeño en el código del programa -agregar o quitar una condición, cambiar alguno de los valores, sumar líneas que impriman más texto, etc.- y que vean cómo cambia el código binario utilizando BARF. Se espera que puedan establecer una relación entre el cambio que realizaron en el código C frente al cambio obtenido en el código binario.

Como cierre, se sugiere repasar los conceptos trabajados durante la clase. Cuando programamos en algún lenguaje de programación necesitamos realizar un paso intermedio para que la computadora pueda ejecutar ese programa ya que ésta sólo sabe ejecutar instrucciones que estén escritas en lenguaje de máquina (**`mov`**, **`cmp`**, etc.). Requerimos entonces un programa especial llamado **compilador** que traduce todo el texto del programa a lenguaje de máquina. Se habla de **código fuente** para referirse a el o los archivos que contienen el programa en algún lenguaje de programación (en nuestro caso sería “`entrada_de_cine.c`”) y de **archivo binario** para referirse al programa ya compilado, en lenguaje de máquina, que puede utilizar directamente la computadora (en nuestro caso sería “`entrada_de_cine`”). Es importante enfatizar que las aplicaciones que utilizamos diariamente suelen estar compiladas: el sistema operativo, las apps que descargamos, el navegador web, el procesador de texto, etc.

Clase 10 - Arquitectura básica de una computadora

En la primera parte de la clase se buscará determinar qué es y qué no es una computadora, y cuáles son sus características distintivas. Para ello, como primera actividad, se propondrá un sencillo juego en donde las y los estudiantes deben “adivinar” cuáles de las siguientes imágenes u objetos[18] son computadoras y cuáles no lo son. El objetivo es que las y los estudiantes comprendan que muchos de los dispositivos electrónicos que utilizamos en nuestra vida cotidiana son computadoras aunque no tengan la forma clásica de una.

Computadora de escritorio		
Computadora portátil		
Smartphone		

[18] Si se pudiera trabajar con objetos físicos a lo largo de toda la clase, tanto los dispositivos como los componentes, mucho mejor. Dependerá de la disponibilidad de cada institución.

Celular		
Mouse		
Consola de videojuegos		
Auriculares		
Impresora		

Cámara fotográfica		
TV LED por dentro		
Cabina del avión Airbus 380		

Finalizada esta primera actividad, el/la docente preguntará a sus estudiantes si se les ocurre qué determina que un artefacto electrónico sea o no una computadora. En este punto puede que se enumeren muchos componentes accesorios (placa gráfica, placa de sonido, disco rígido, placa madre, distintos periféricos, etc.). El/La docente los irá anotando en el pizarrón y, en caso de que no se mencionen el procesador y la memoria, buscará promover la aparición de estos componentes con frases como:

- Si abrimos muchos programas al mismo tiempo la computadora empieza a andar lenta porque se quedó con poca … (memoria *RAM*)
- Una computadora puede ser más rápida que otra porque tiene un … más rápido (*procesador*)

Si no se hubieran mencionado periféricos de entrada (teclado, mouse, micrófono, escáner), de salida (monitor, impresora, auriculares) y de entrada/salida (pantalla táctil, impresora multifunción) el/la docente completará la lista con la categoría que faltase.

Al concluir con el listado de componentes de una computadora, el/la docente seleccionará aquellos que sean periféricos de entrada y preguntará a la clase si encuentran alguna relación

entre ellos, una característica común, si podrían agruparlos dentro de alguna categoría. Entre las características que podrían ser mencionadas destacamos:

1. Se conectan a la “computadora”[19]
2. Sirven para que la computadora reciba información del mundo
3. Permiten a las personas ingresar datos

A medida que se vayan mencionando las características anteriores se puede ir realizando un gráfico en el pizarrón en el que haya un cuadrado vacío que represente a la computadora y, una vez mencionada las características 1 y 2 o 1 y 3, flechas que conecten a cada uno de los periféricos con la computadora. De este modo comenzaremos a construir la figura 10.1.

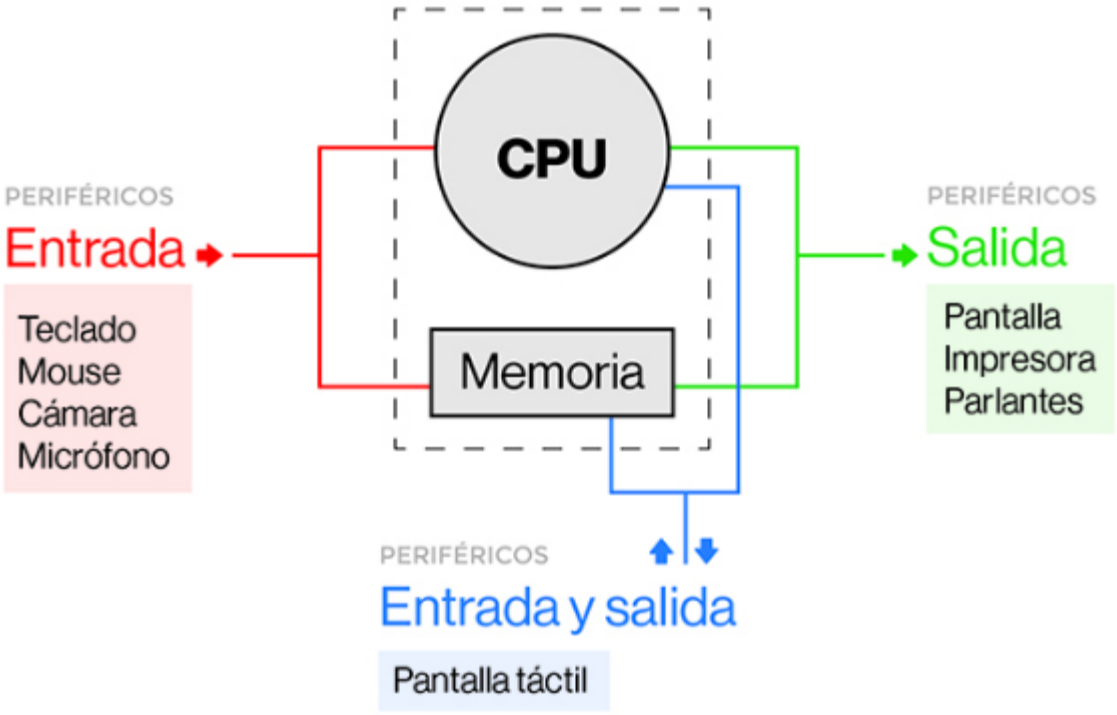


Figura 10.1

Luego de caracterizar y categorizar a los periféricos de entrada se procederá de modo similar con los periféricos de salida -colocando flechas que vayan de la computadora al periférico- y los de entrada/salida -colocando flechas bidireccionales entre la computadora y el periférico.

Entre los componentes listados puede que haya también periféricos de almacenamiento (disco rígido, memoria USB, CD) y periféricos de comunicación (módem, router, placa de red). Se puede mencionar que estos también son periféricos.

[19] *Computadora* se menciona entre comillas puesto que todavía no se llegó a una definición acerca de qué es una computadora.

Una vez colocadas todas las flechas de los periféricos se puede intentar reconocerlos en otros dispositivos que no sean la computadora de escritorio o las notebooks y netbooks. Por ejemplo, ¿qué periféricos tenemos en un smartphone? ¿Y en una consola de videojuegos? ¿Y en la máquina para votar? Por ejemplo, los parlantes de una computadora portátil o el teclado de un celular táctil vienen embebidos. Sin embargo, no dejan de ser periféricos de la computadora. Simplemente están integrados en ella.

La cuestión que resta abordar es a dónde entran y de dónde salen esas flechas que conectan a los periféricos. ¿Se puede ser más específico que simplemente notando que se conectan con la computadora? Entre los componentes listados hay dos que todavía no fueron incorporados en el esquema: el procesador y la memoria RAM. Se les puede preguntar a la clase qué creen o suponen que realizan estos dos componentes y en qué parte del esquema los ubicarían.

Algunas características de la memoria RAM que pueden surgir:

- Sirve para guardar información.
- Es una memoria volátil, es decir, si se corta la corriente se pierden los datos.
- Los programas se cargan en la memoria RAM.
- Si te queda poca o nula memoria RAM la computadora anda lenta.

Algunas características del procesador que pueden surgir:

- Realiza todos los cálculos.
- Hay distintas velocidades de procesadores.
- Una computadora puede tener muchos procesadores.
- Suele generar mucho calor.

El/La docente, para relacionar estos componentes con lo visto durante la primera parte de la materia, puede proponer pensar qué pasaría si nuestra computadora no tuviera procesador o no tuviera memoria RAM. En el primer caso, por ejemplo, no podríamos crear programas ya que es el procesador el que ejecuta las instrucciones (en las clases 13 y 14 se verá este aspecto con más detalle). Si no tuviéramos memoria RAM nuestros programas no podrían guardar información en variables ya que éstas se almacenan en la memoria RAM (en las clases 11 y 12 se profundizará su funcionamiento). Por lo tanto, una máquina que no nos permitiera correr programas o que no les permitiera a los programas guardar información en tiempo de ejecución, no nos permitiría hacer muchas cosas. Por lo tanto, para que una máquina sea considerada una computadora debe tener al menos:

- Al menos un procesador.
- Memoria RAM.
- Al menos un periférico de Entrada o un periférico de Salida[20].

[20] Este último requisito no es técnicamente indispensable pero si no se lo tiene en cuenta la utilidad de la computadora en cuestión se ve francamente disminuida.

Para finalizar, resta ver más específicamente a dónde se conectan los periféricos. ¿A la RAM o al procesador? Probablemente las y los estudiantes supongan que al procesador que es el encargado de ejecutar las instrucciones. Esto es correcto pero además algunos dispositivos utilizan la memoria RAM. Por ejemplo, la impresora, para imprimir un archivo, necesita recuperarlo de la memoria RAM[21].

---

[21] Más detalles sobre el tema pueden consultarse en https://es.wikipedia.org/wiki/Acceso_directo_a_memoria.

Clase 11 - Almacenamiento y memoria I

Proponer un intercambio inicial con las y los estudiantes con el objetivo de indagar cuáles son sus conocimientos previos acerca de la “memoria de la computadora”. Una primera pregunta lo suficientemente abierta puede ser:

- ¿Cuánta memoria tiene su computadora? ¿Y su celular? ¿Tiene memoria el celular?

Probablemente la mayoría pueda decir con mayor o menor grado de seguridad cuánta memoria tiene su computadora o su celular pero muy probablemente se refieran a la memoria RAM o, indistintamente, a la memoria RAM y a la capacidad del disco rígido o la tarjeta de memoria. Aquí se puede tratar de guiar a las y los estudiantes para que reconozcan que hay al menos 2 tipos diferentes de memoria con preguntas como:

- Cuando escribo un documento en la computadora o bajo una foto/video o instalo un programa, ¿en qué parte de la computadora queda guardado?
- ¿Nunca les pasó que un programa les pidiera que guarden sus trabajos, sesiones, juegos, etc. antes de salir? Si los están viendo y modificando ¿no están almacenados en algún lugar? ¿Por qué se pierde esa información si no decimos que queremos guardarla?
- ¿Alguna vez escucharon hablar de memoria RAM? ¿Y de disco rígido?

Luego de una breve discusión sobre estas preguntas, se espera que se reconozca que existen 2 clases de memoria. Si no hubiera surgido ya en el debate, el/la docente contará que la memoria RAM es una memoria de trabajo en donde se guarda información de manera temporal: cuando se corta el suministro eléctrico, esa información se pierde, es volátil. Por otro lado, el disco rígido almacena información de forma permanente o persistente. Entonces:

- ¿Dónde se guarda lo que estoy escribiendo en un archivo de texto si todavía no apreté la opción de “guardar”? Y, ¿dónde se guarda ese mismo archivo si le pongo un nombre y aprieto la opción de “guardar”?

Hasta aquí las y los estudiantes reconocerán que hay 2 tipos de memoria: una volátil y la otra persistente o permanente. Con el objetivo de que comprendan por qué no usamos siempre el disco rígido, se les propondrá que busquen en Internet cuánto sale una placa de memoria y cuánto un disco rígido. Se les pedirá además que anoten la capacidad de almacenamiento de ambos. A continuación se hará una puesta en común anotando ambas características en el

pizarrón. Muy probablemente la capacidad de la memoria RAM esté expresada en GB y la del disco rígido en GB o TB.

En este punto se aprovechará para comprender las unidades de almacenamiento. Como actividad, se les pedirá que creen un archivo vacío en Linux (botón derecho del mouse) y que se fijen cuánta pesa (0 Byte). Luego, que escriban una sola letra en el archivo, lo guarden y noten cuánto pesa (unos pocos Bytes dependiendo el carácter que hayan escrito). A continuación se les pedirá que copien 1000 veces seguidas (en la misma línea) esa letra, guarden el archivo y verifiquen cuánto pesa (1 KB aproximadamente). Por último deberán estimar cuántas letras deberían escribir para generar un archivo que pese 1 MB y un archivo que pese 1 GB.

En la puesta en común se explicitará que el Byte es una unidad de almacenamiento muy pequeña que permite guardar poca información y se construirá en conjunto la siguiente tabla:

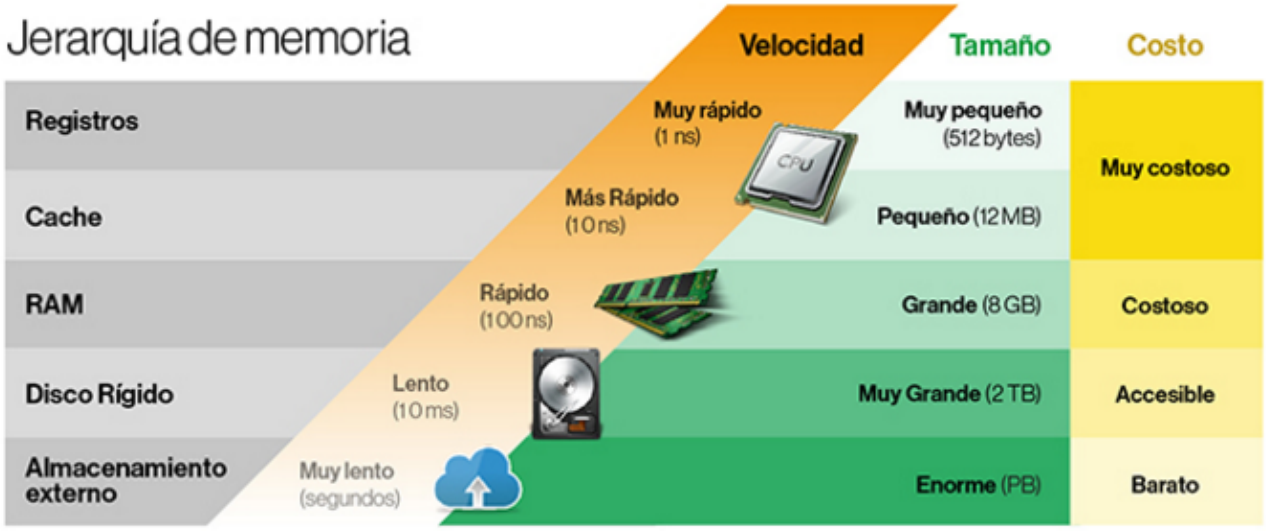
1 KB	~1000 Byte
1 MB	~1000 KB
1 GB	~1000 MB
1 TB	~1000 GB

Al finalizar esta pequeña actividad, se retomará la discusión anterior sobre el precio y capacidad de almacenamiento de las memorias RAM y los discos rígidos. Se pondrá de manifiesto que la memoria RAM es mucho más cara por cada GB que el disco rígido y se les preguntará a las y los estudiantes:

> - ¿Por qué no usamos solamente el disco rígido entonces, si es más barato?

El/La docente guiará la discusión buscando que las y los estudiantes piensen qué dimensiones resultan más importantes a la hora de comparar 2 computadoras, para así fomentar que surja la noción de “velocidad” o “rapidez”.

Aquí el/la docente realizará una breve explicación de que la mayor diferencia entre los distintos tipos de memoria es el **tiempo de acceso**, cuanto más rápida más cara y más cerca físicamente del procesador, que procesa la información que allí se guarda (se profundizará esta relación en la clase 14). Se mostrará una imagen similar a la siguiente para conceptualizar la jerarquía:



Sobre la memoria caché se trabajará en la clase 12 y los registros del CPU serán mencionados en la clase 14.

Para la siguiente actividad se les pedirá a las y los estudiantes que indiquen y justifiquen en qué memoria (RAM o disco) se guardan los siguientes recursos:

1. Una **variable** como las que definieron en Alice y en C.
2. Un **programa** en un archivo .c que hayan hecho en un editor de C.
3. Un **programa** en C **que** están escribiendo en el editor y aún **no guardaron**
4. Un **programa** en un archivo .c **que tienen abierto** en el editor.

(En los puntos 1 y 3 la respuesta es memoria RAM mientras que en el punto 2 es disco rígido. El punto 4 es interesante porque sucede que el archivo se guardó en disco pero cuando se lo abre en el editor también se carga en memoria RAM para poder trabajar más rápido con él.)

En la puesta en común sobre las respuestas, el/la docente hará hincapié en el hecho de que las variables se guardan en la memoria RAM y preguntará:

- ¿Cómo encuentra la computadora dónde está guardada la variable?
- Si tenemos 8GB de memoria RAM y nuestra variable ocupa 4 Bytes, ¿cómo hace para buscar rápidamente esos 4 Bytes entre los 8 mil millones de Bytes existentes?

Con estas preguntas se busca motivar la idea de que la memoria (de cualquier tipo) está ordenada a través de posiciones o direcciones: 1, 2, 3, etc. Por lo tanto, para saber cuál es el valor de una variable debemos saber antes en qué posición está guardada. ¿Cómo realiza este proceso la computadora? Se verá con más detalle en la clase 14. Se mostrará una imagen similar a la siguiente para clarificar:

Para reforzar las unidades de almacenamiento se les pedirá a las y los estudiantes que estimen cuál sería el valor de max si nuestra memoria fuera de 2 GB.

Como actividad práctica, se les pedirá a las y los estudiantes que indaguen las características de la memoria RAM y la capacidad de disco que tienen en sus computadoras y sus celulares.

Por último, el/la docente remarcará los aspectos más importantes trabajados en esta clase:

- Reconocer que en las computadoras que usamos (netbooks, celulares, etc.) hay dos tipos de memoria, una volátil (RAM) y una persistente (disco rígido, tarjeta de memoria).
- Entender al tiempo de acceso como el criterio para diferenciar la velocidad de distintos tipos de memoria, es decir, el tiempo que le demanda a una memoria leer o escribir un dato.
- Notar que las variables de nuestros programas se almacenan en la memoria RAM, la cual, para saber dónde guardó cada dato, está dividida en direcciones. A cada dirección o posición de memoria RAM corresponder un valor almacenado en ese lugar.
- Familiarizarse con las distintas unidades que se utilizan en computación para referirse al tamaño de los datos almacenados: Byte, KB, MB, GB, TB, PB, etc.

Clase 12 - Almacenamiento y memoria II

Como primera actividad, las y los estudiantes se dividirán en grupos y a cada grupo se les darán las siguientes 3 secuencias de acciones que un programa de edición realiza con una foto almacenada en una memoria USB o pendrive:

Secuencia 1	Secuencia 2	Secuencia 3
abrir(“F:\fotaza.png”) aplicarFiltroBN	abrir(“F:\fotaza.png”) aplicarFiltroBN guardarFoto	abrir(“F:\fotaza.png”) aplicarFiltroBN guardarFoto autoRecortar guardarFoto

Las instrucciones utilizadas realizan las siguientes acciones:

- **abrir(“F:\fotaza.png”)**: abre la foto “fotaza.png” guardada en la unidad extraible F, es decir, la memoria USB o pendrive.
- **aplicarFiltroBN**: convierte la “fotaza” en una foto Blanco y Negro.
- **guardarFoto**: guarda la foto con los cambios aplicados en la memoria USB.
- **autoRecortar**: quita los bordes de la imagen.

El objetivo del ejercicio es calcular cuánto tardará la ejecución de cada secuencia, considerando el tiempo que demora ejecutar una instrucción y el tiempo que se requiere para leer o escribir la foto en la memoria USB. Para ello, cada grupo recibirá alguno de los siguientes tiempos propuestos:

A	- Ejecutar una instrucción: 1 ns - Leer o escribir en la memoria USB: 1 ns
B	- Ejecutar una instrucción: 1 ns - Leer o escribir en la memoria USB: 10 ns
C	- Ejecutar una instrucción: 1 ns - Leer o escribir en la memoria USB: 100 ns

Mientras los grupos se ponen a resolver la actividad, el/la docente irá pasando por las mesas de trabajo para irlos orientando. Cuando todos hayan terminado la actividad se realizará una puesta en común y se compararán los tiempos para cada una de las secuencias considerando los 3 pares de tiempos posibles, como se muestra en la siguiente tabla:

	Secuencia 1	Secuencia 2	Secuencia 3
A	(1+1) + 1 = **3**	(1+1) + 1 + (1+1) = **5**	(1+1) + 1 + (1+1) + 1 + (1+1) = **8**
B	(1+10) + 1 = **12**	(1+10) + 1 + (1+10) = **23**	(1+10) + 1 + (1+10) + 1 + (1+10) = **35**
C	(1+100) + 1 = **102**	(1+100) + 1 + (1+100) = **203**	(1+100) + 1 + (1+100) + 1 + (1+100) = **305**

El color azul indica el tiempo que demanda ejecutar una instrucción, el rojo el tiempo que requiere leer o escribir en la memoria USB y el negro el tiempo total de realizar todas las acciones de una secuencia.

En la tabla se aprecia que (a igual tiempo de ejecución de una instrucción) el tiempo total que toma ejecutar una secuencia aumenta muchísimo en función de la cantidad de accesos a la memoria USB y cuanto más costosos (temporalmente) resulten estos. Más aún, el tiempo de ejecución de las instrucciones es despreciable en comparación con el tiempo que toma acceder tan solo una vez a la memoria USB.

El/La docente contará que el escenario actual de las computadoras que usamos se parece más bien al caso C, es decir, capacidad para ejecutar muy rápido una instrucción pero con memorias USB muy lentas en comparación, que impactan fuertemente en la performance. Lo mismo ocurre con los discos duros. Aquí se podría mostrar una tabla con diferentes tiempos de acceso para memorias RAM, discos duros y memorias USB.

Se pueden usar las siguientes preguntas como disparadoras:

- ¿A alguien se le ocurre cómo se podría resolver este problema?
- ¿Tendrá una solución?
- Cuando ejecutan la instrucción **guardarFoto**, ¿dónde creen que se guarda?
- Si retiran la memoria USB sin “quitar hardware con seguridad”, ¿los cambios realizados en la foto quedan guardados en el pen drive?
- ¿Alguien escuchó o leyó la palabra *caché* alguna vez?

El/la docente describirá brevemente la secuencia de pasos que se realizan al guardar un dato en *caché*:

- La foto se carga (se copia) en la memoria RAM, que es más rápida que la memoria USB.
- Las instrucciones modifican la foto guardada en la memoria RAM.
- **guardarFoto** salva la foto en la memoria RAM, no en la memoria USB.
- Al quitar el pendrive de manera segura todos los cambios realizados se guardan allí.
- Si no se quitara de forma segura los cambios no se guardarían en la memoria USB y se perderían.

De este modo, sólo se accede una única vez a la memoria USB para leer el archivo y una única vez para guardarla, disminuyendo drásticamente el tiempo que requiere realizar una secuencia de instrucciones que lean y escriban muchas veces en dicha memoria.

Aquí se les puede preguntar a las y los estudiantes cuáles creen que son las ventajas y desventajas de esta estrategia. A modo de ejemplo, se enumeran algunas características:

- Las instrucciones se ejecutan más rápido ya que la memoria RAM es mucho más rápida que la memoria USB.
- Si se retira el pendrive sin quitar de manera segura los cambios no se guardan.
- Si se apaga la compu (por corte de luz, se queda sin batería, etc.) los cambios no se guardan.
- No se pueden *cachear* grandes volúmenes de datos ya que la memoria que funciona como *caché* suele ser más pequeña que la que almacena los datos.

A modo de síntesis, el/la docente les contará a las y los estudiantes que así como los datos de la memoria USB se guardaban temporalmente en la memoria RAM para realizar las operaciones de forma más rápida, esta misma idea se lleva a cabo en numerosas situaciones:

- Los datos del disco rígido se cargan en la memoria RAM.
- Los datos de la memoria RAM se cargan en unas memorias especiales, más rápidas que las RAM, a las que se las conoce como memorias *caché* L1, L2, L3, etc.
- Los datos de la memoria *caché* se guardan en unas memorias más rápidas aún conocidas como registros.
- Algunos datos de navegación se guardan en la computadora para no tener que volver a conseguirlos a través de internet. Por ejemplo, las imágenes de una página se suele guardar en la computadora para que cuando volvamos a acceder a esa misma página, la misma se cargue más rápido puesto que las imágenes ya las tenemos.

Uno de los problemas que se presentan es que la memoria *caché* tiene espacio limitado y generalmente muy pequeño. No se puede guardar todo en la *caché*. Lo que faltaría comprender es qué ocurre cuando ésta se llena y queremos almacenar un nuevo dato. ¿Cuál sacamos para poder guardar el nuevo dato?

Se les contará a las y los estudiantes cómo se modificaría la memoria *caché* considerando un algoritmo en particular y suponiendo que al comenzar no hay ningún dato. El/La docente explicará brevemente el algoritmo LRU (*Least Recently Used*, en inglés, o Menos Recientemente Usado, en español) el cual lleva un registro del tiempo de vida de cada dato que se va agregando a la *caché* y cuando ésta se llena y se requiere guardar uno nuevo, retira el dato con mayor tiempo de vida y

agrega al nuevo. Por lo tanto, el algoritmo LRU debe saber qué dato se halla en cada lugar o posición de memoria de la *caché* y hace cuánto lo tiene colocado en ese lugar. En caso de que se pidiera un dato que ya está en *caché* se reinicia su tiempo de vida.

Como cierre, se propone realizar un repaso de los puntos más importantes trabajados durante la clase. Una primera idea que se abordó es que el procesador puede ejecutar instrucciones en un tiempo mucho menor del que se tarda en leer o escribir un dato en la memoria RAM, el disco rígido o una memoria extraíble. Por lo tanto, si no se tomara ninguna medida, la velocidad que demandaría ejecutar un programa dependería de cuán rápida es la memoria que estamos usando, a pesar de que la CPU que tengamos sea sumamente veloz. Para resolver este problema, se suele utilizar una memoria más pequeña y mucho más rápida a la que se la llama memoria *caché*, en la cual se guardan los datos que se utilizan con mayor frecuencia. Una de las principales ventajas es que ayuda a disminuir el tiempo que toma ejecutar un programa. Como desventaja, al ser una memoria muy pequeña, podemos almacenar pocos datos a la vez por lo que se debe tener un criterio de reemplazo que permita desalojar un dato que está en *caché* para guardar uno nuevo. Uno de los algoritmos utilizados es el LRU, que reemplaza el dato que menos frecuentemente se haya estado accediendo.

➢ *Anexo*

En caso de que quedara tiempo durante la clase, se pueden comentar las siguientes dos analogías para reforzar el concepto de *caché*:

1. La mochila o bolso que las y los estudiantes llevan a la escuela contiene un pequeño conjunto de cosas que ellas y ellos consideran que van a utilizar con frecuencia o que necesitan tener a mano. De este modo, al tenerlas en la mochila (*caché*) no es necesario volver a la casa (*RAM*) cada vez que se necesita alguno de esos objetos. En la analogía, la mochila, al igual que la *caché*, tiene menos capacidad que la casa, es decir la RAM, es de rápido acceso y se usa para guardar objetos necesarios en el futuro próximo que estaban originalmente en el hogar.
2. En las bibliotecas, las y los bibliotecarios a cargo suelen tener en una estantería muy cerca de ellos los libros que se suelen pedir con frecuencia. De este modo, las y los bibliotecarios minimizan el tiempo total de ir a buscar los libros ya que la mayoría de los pedidos los tienen a la mano. En este caso, la estantería cercana sería la *caché*, la cual tiene un tamaño mucho menor al de toda la biblioteca, es de rápido acceso y se usa para guardar los libros más pedidos en el último tiempo, los cuales estaban originalmente en estantes más alejados.

Por último, como actividad para el hogar, se propone la siguiente actividad para determinar cómo evolucionaría en el tiempo el estante *caché*, considerando que al inicio se encuentra vacío.

Suponiendo que contamos con un estante *caché* con 3 lugares para guardar libros:

- ¿Cómo evolucionarían en el tiempo los libros que va almacenando para cada una de las siguientes tres secuencias de pedidos?
- Y si el estante tuviera 4 lugares, ¿qué cambiaría?

Secuencia 1	Secuencia 2	Secuencia 3
Rayuela	Rayuela	Rayuela
Los versos del capitán	Ulises	Los versos del capitán
Rayuela	Los versos del capitán	Ulises
Ulises	Relato de un náufrago	Relato de un náufrago
Relato de un náufrago	Rayuela	Rayuela
El nombre de la rosa	El nombre de la rosa	Los versos del capitán
Ulises	Ulises	Ulises

Luego se realizará una puesta en común en donde se discutirán las respuestas y se debatirá acerca de si el algoritmo es infalible o no. Se puede notar que en la secuencia 3 el hecho de tener un estante *caché* de 3 estantes y utilizar el algoritmo LRU no resultó de utilidad puesto que cada libro que se iba pidiendo no estaba en *caché* y había que irlo a buscar a su estante lejano original. ¿Se podría pensar un algoritmo que funcionara mejor para este caso en particular? Un algoritmo que funciona bien para este caso es LIFO (*Last in, first out*, en inglés, o “último en entrar, primero en salir”, en español), por ejemplo. Sin cambiar el algoritmo LRU utilizado y agregando un estante más, es decir, con 4 estantes este problema desaparece.

Resulta interesante notar el impacto que tienen los fallos (*misses*) en la predicción de los algoritmos, es decir, cuando un libro no lo tenemos en el estante cercano y hay que ir a buscarlo a los estantes más lejanos de la biblioteca. Si se diera un caso en donde el algoritmo nunca predijese correctamente los libros más frecuentes (como sucedía con la secuencia 3), entonces tener un estante *caché* sería igual a no tenerlo. Por lo tanto, los buenos algoritmos de *cacheo* son aquellos que dado un contexto de uso predicen con una tasa alta de aciertos los datos que se van a necesitar de la memoria RAM.

Como tarea e instancia de evaluación se les pedirá a las y los estudiantes que piensen otra política de reemplazo y que justifiquen en qué casos funcionaría bien y en cuáles no.

Como cierre, se puede volver a mostrar el gráfico de la jerarquía de memorias que vieron en la clase pasada y notar la relación capacidad de almacenamiento/velocidad entre disco rígido, memoria RAM y memoria *caché*.

Solución del ejercicio de LRU

En las siguientes tablas, cada fila muestra cómo se modificaron los estantes frente a un nuevo pedido y el número entre paréntesis indica el tiempo de vida de cada libro. Sólo se muestra la evolución temporal para un estante *caché* de 3 lugares. Notar que si un libro está en el estante sin ser pedido, va aumentando su tiempo de vida a medida que pasa el tiempo.

Evolución temporal para la secuencia 1:

Tiempo	Lugar 1	Lugar 2	Lugar 3
1	Rayuela (1)	-	-
2	Rayuela (2)	Los versos del capitán (1)	-
3	Rayuela (1)	Los versos del capitán (2)	-
4	Rayuela (2)	Los versos del capitán (3)	Ulises (1)
5	Rayuela (3)	Relato de un náufrago (1)	Ulises (2)
6	El nombre de la rosa (1)	Relato de un náufrago (2)	Ulises (3)
7	El nombre de la rosa (2)	Relato de un náufrago (3)	Ulises (1)

Evolución temporal para la secuencia 2:

Tiempo	Lugar 1	Lugar 2	Lugar 3
1	Rayuela (1)	-	-
2	Rayuela (2)	Ulises (1)	-
3	Rayuela (3)	Ulises (2)	Los versos del capitán (1)
4	Relato de un náufrago (1)	Ulises (3)	Los versos del capitán (2)
5	Relato de un náufrago (2)	Rayuela (1)	Los versos del capitán (3)
6	Relato de un náufrago (3)	Rayuela (2)	El nombre de la rosa (1)
7	Ulises (1)	Rayuela (3)	El nombre de la rosa (2)

Evolución temporal para la secuencia 3:

Tiempo	Lugar 1	Lugar 2	Lugar 3
1	Rayuela (1)	-	-
2	Rayuela (2)	Los versos del capitán (1)	-
3	Rayuela (3)	Los versos del capitán (2)	Ulises (1)
4	Relato de un náufrago (1)	Los versos del capitán (3)	Ulises (2)
5	Relato de un náufrago (2)	Rayuela (1)	Ulises (3)
6	Relato de un náufrago (3)	Rayuela (2)	Los versos del capitán (1)
7	Ulises (1)	Rayuela (3)	Los versos del capitán (2)

Clase 13 - CPU I

Proponer a las y los estudiantes que descubran qué procesador tienen en su máquina, tablet y/o celular. Deben registrar marca, modelo, velocidad y cantidad de procesadores. El/la docente irá pasando por las mesas ayudando a que las y los estudiantes encuentren una herramienta que les permita determinar estos valores.

Al finalizar esta primera actividad, el/la docente anotará en el pizarrón las características de los procesadores que encontraron y se fomentará la siguiente discusión motivada por preguntas como, por ejemplo:

- ¿Cuántas marcas de procesadores distintas recolectamos? Probablemente sean 1 o 2 empresas. En este punto se puede hacer una pequeña digresión acerca de lo concentrado del mercado en relación a la producción de todos los procesadores que usamos en nuestra vida cotidiana y dejar para pensar la pregunta: ¿qué implicancias podría tener esta situación?
- ¿Cuál es la velocidad mínima y cuál la máxima? ¿Cómo se mide la velocidad? ¿Qué significa GHz?

En este punto razonar junto con las y los estudiantes las dos componentes de la unidad “GHz”: Giga y Hertz. Para empezar tratar de que recuperen la noción de Giga, que ya la vieron en la clase 11. Luego, indagar si las y los estudiantes reconocen el término Hz de algún otro lado (materias como Física o mismo la radio FM) y observar que mide “cantidad de «algo» por unidad de tiempo”.

- ¿Qué es ese “algo” en el caso del procesador? ¿Qué piensan que nos interesa saber sobre un algo que ocurre muchas veces por segundo en la computadora?

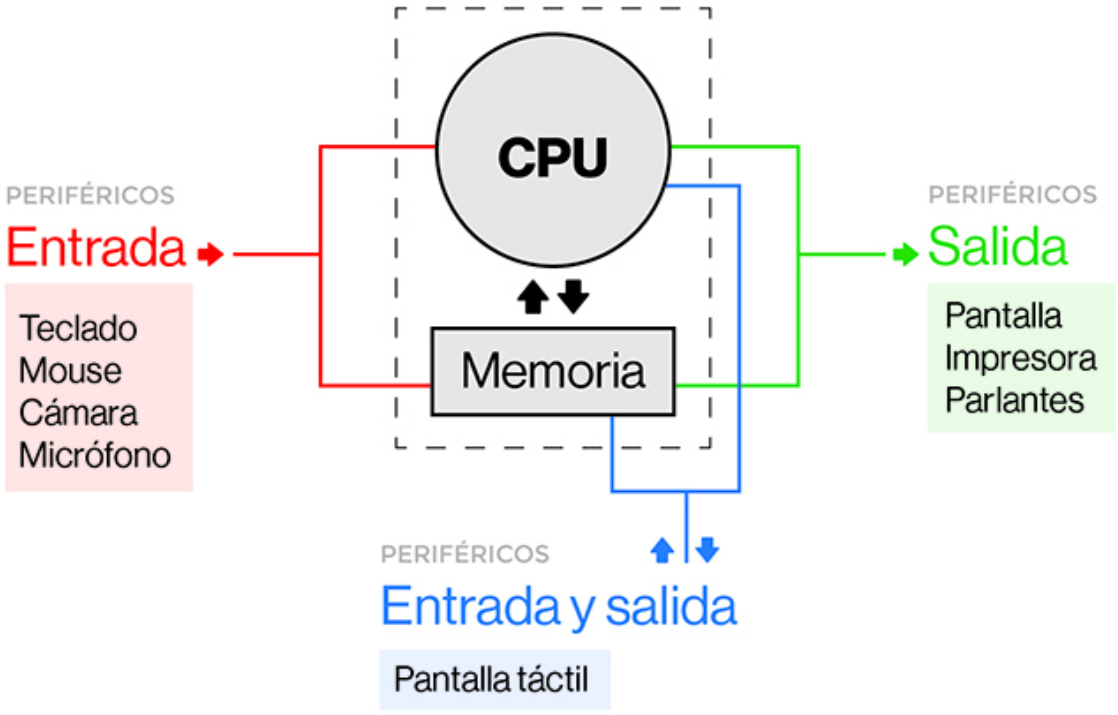
En esta discusión, el/la docente puede recapitular lo hecho en la clase 9 cuando desensamblaron un programa y vieron las instrucciones que ejecuta la máquina. En esa clase una de las conclusiones fue que “la computadora lo único que sabe hacer es seguir una tras otra ese tipo de instrucciones: el código de máquina”.

- Entonces, ¿cuál podría ser uno de los factores que nos muestren que una computadora es más rápida que otra?

Llegado este punto se espera que las y los estudiantes destaquen la cantidad de instrucciones que puede ejecutar una computadora por segundo, enfatizando que es el procesador el que las ejecuta.

Clase 14 - CPU II

Como primera actividad se buscará reconstruir entre todos el diagrama de alto nivel realizado en la clase 10 sobre qué partes componen una computadora y cómo se interrelacionan, haciendo foco principalmente en la relación entre CPU y memoria RAM, y no tanto en los periféricos.



Al concluir el esquema, se mostrará el programa en lenguaje ensamblador que habían obtenido al desensamblar (clase 9) el programa C que habían hecho en la clase 8 y se recordará la conclusión de la clase pasada: “la CPU lo único que sabe hacer es ejecutar una instrucción tras otra a una velocidad muy rápida: miles de millones por segundo”.

A esta altura las y los estudiantes ya deberían tener una noción de qué hace un procesador pero resta saber cómo lo hace. Para ello, primero se buscará que las y los estudiantes ensayen algunas propuestas e ideas fomentando la discusión con preguntas como:

- ¿Cómo hace la CPU para saber qué instrucción tiene que ejecutar en cada instante?
- ¿Cómo hace para sumar, restar, multiplicar, dividir 2 números?
- ¿La CPU cuenta con todos los datos necesarios para ejecutar todas las instrucciones que queremos? Por ejemplo, si le queremos sumar 1 al valor de una variable, ¿dónde se encuentra dicho valor? ¿En la CPU? Esta última pregunta será útil para recordar que las variables se guardan en memoria y, por ende, no están en la CPU, como se vio en la clase 4 y la clase 11.

El/la docente mostrará en el pizarrón o la pantalla un programa muy sencillo en pseudo-ASM[22] como por ejemplo:

```
inicio:
    MOV   EAX, #6345
    MOV   EBX, #6349
    ADD   EAX, EBX
```

Propondrá a las y los estudiantes que hagan las veces de CPU. Primero deberán indicar por dónde comenzar. Es probable que el título de la etiqueta “inicio” condicione que elijan empezar por la instrucción “MOV EAX, #6345”. Se le preguntará a las y los estudiantes qué creen que son cada una de las partes de la instrucción y qué hace. Lo primero que se motivará es que “MOV A, B” lee el valor que está en B y lo escribe en A. Luego, se verá que #6345 es una dirección de la memoria en donde se guarda un valor, como observaron en la clase 11. Aquí se puede usar el esquema que vincula la CPU con la Memoria para mostrar que la CPU tiene que ir a buscar ese dato a la Memoria. Por último, el/la docente comentará que EAX es otro tipo de componente destinado al almacenamiento de datos llamado “registro”, que permite guardar un sólo valor por vez pero que es mucho más rápido que la memoria y que está dentro de la CPU, como vieron en las clases 11 y 12.

En este punto, es esperable que las y los estudiantes ya puedan leer y entender el programa completo. Probablemente puedan surgir como dudas qué hace el ADD (suma 2 números) y por qué se usa el # antecediendo a los números 6345 y 6349 (es la nomenclatura elegida para saber que nos referimos a una dirección de memoria, es decir, que #1234 no hace referencia al valor "1234" si no al que se encuentra en la dirección de memoria 1234).

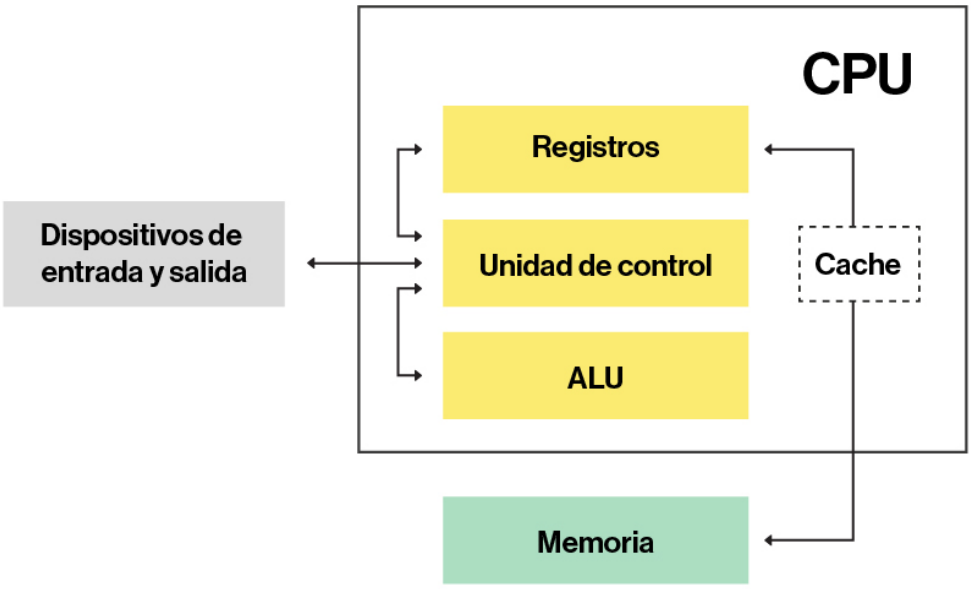
Llegado este punto se concluirá colectivamente qué hace el programa (sumar los valores que se encuentren en las direcciones 6345 y 6349 y dejar el resultado en un registro).

Al concluir esta actividad, el/la docente resumirá qué fue lo que hicieron:

1. Leer una instrucción.
2. Interpretar qué hace, cuántos valores toma y si hay que buscar alguno de ellos en la Memoria.
3. Ejecutar la instrucción.
4. Volver a 1.

El/la docente bajará un nivel más de abstracción y mostrará en el pizarrón o en la pantalla un diagrama interno de la CPU en donde figuren: los registros, la Unidad de Control y la ALU (Unidad Aritmético-Lógica).

[22] ASM es abreviatura de lenguaje ensamblador.



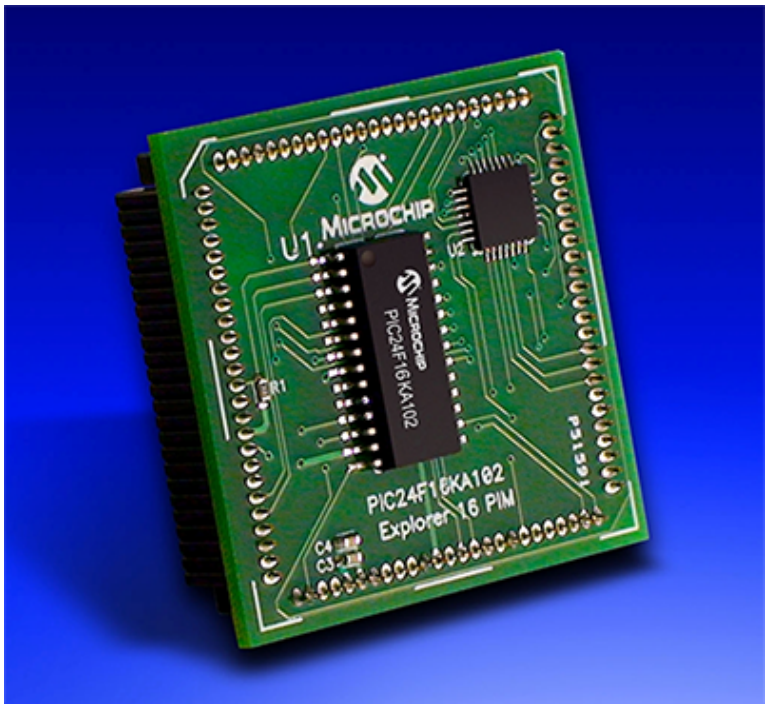
Con el diagrama a la vista, se recomienda repasar el ciclo de instrucción para el segundo MOV y el ADD con el objetivo de que las y los estudiantes noten que:

5. la Unidad de Control decodifica la instrucción, chequea cuáles son sus parámetros y dónde están (registros o Memoria), y, básicamente, determina el flujo del ciclo;
6. los registros sirven para almacenar información y poderla acceder muy rápidamente;
7. la ALU hace las cuentas.

Como actividad entregable para que las y los estudiantes practiquen, se les pedirá que escriban todos los pasos que debe realizar la computadora para poder ejecutar el siguiente programa:

```
sumar2numeros:
        ADD    EAX, #2385
```

Para terminar de bajar en el nivel de abstracción se mostrará un microchip por dentro (puede ser uno real o una imagen) y se observará que todos los diagramas que estuvieron haciendo son una manera de abstraer cómo funcionan las cosas, un diseño, distintas capas para comprender y analizar el mismo objeto.



Como conclusión final es importante que las y los estudiantes comprendan que en esencia, una computadora no es más que la interconexión de muchos componentes electrónicos por los que circula electricidad. Y que cuando hablamos de código binario, como vieron en la clase 9, era una forma de abstraer que estábamos refiriendo al alto voltaje (1) y bajo voltaje (0) que se transmite a través de los circuitos.

Clase 15 - Relación hardware-software

En esta clase se realizará una síntesis de los temas trabajados hasta el momento. Se dividirá en dos instancias, un breve repaso oral y un trabajo escrito que deberá ser entregado al final de la clase.

Para comenzar, el/la docente propiciará un intercambio breve para retomar los conceptos de software y hardware y los componentes vistos hasta el momento:

- ¿Cómo sería la computadora sin software?
- ¿Qué es el software?
- ¿Cómo sería la computadora sin hardware?
- ¿Qué es el hardware?
- ¿Qué componentes de hardware vimos en clase? ¿De qué se ocupa cada uno?

Se espera que se destaque la diferencia entre los componentes físicos, tangibles y los programas y aplicaciones. También es importante entender a ambas partes como constitutivas de la computadora. Por último, un breve repaso de los componentes facilitará el desarrollo de la actividad que el/la docente presentará a continuación.

El/la docente mostrará el siguiente extracto de un programa escrito en el lenguaje C:

```
imprimir_un_texto("Ingrese su edad:");
int edad = leer_entero();

if (edad < 19) {
      imprimir_un_texto("Ud. es un joven.");
} else {
      if (edad < 65) {
            imprimir_un_texto("Ud. es un adulto.");
      } else {
            imprimir_un_texto("Ud. es un adulto mayor.");
      }
}

imprimir_un_texto("El año que viene Ud. cumplirá:");
imprimir_un_numero(edad + 1);
```

Las y los estudiantes deberán realizar por escrito un trabajo con la siguiente consigna:

1. Describir qué hace el programa.
2. Imaginar una computadora que no cuenta con cada componente de la lista y describir qué ocurriría si se intentara ejecutar el programa en cada caso:
    a. Computadora sin **CPU**
    b. Computadora sin **unidad de control**
    c. Computadora sin **ALU**
    d. Computadora sin **memoria RAM**
    e. Computadora sin **memoria *caché***
    f. Computadora sin **disco**
    g. Computadora sin **compilador**
    h. Computadora sin **teclado**
    i. Computadora sin **luz ni batería**
    j. Computadora sin **monitor o pantalla**

A continuación, se brindan algunas respuestas correctas posibles para cada ítem y con qué clases previas se relacionan.

1. Descripción del programa:
    El programa le pregunta al usuario cuál es su edad y:
    - si tiene 18 años o menos le dice que es un joven,
    - si tiene entre 19 y 64 le dice que es un adulto y
    - si tiene 65 años o más le dice que es un adulto mayor.

    Por último, el programa te dice cuántos años cumplirás el año siguiente.

2. Las respuestas obtenidas deberían ser similares a las siguientes:
    a. Computadora sin **CPU:**
        Una computadora sin CPU no tendría forma de ejecutar las instrucciones, como se vio en las clases 10, 13 y 14.
    b. Computadora sin **unidad de control:**
        Una computadora sin unidad de control no podría decodificar las instrucciones ni chequear cuáles son y dónde se encuentran sus parámetros. Este tema fue abordado en la clase 14.
    c. Computadora sin **ALU:**
        Una computadora sin ALU no tendría forma de realizar operaciones aritméticas, como cuando el programa incrementa en un año la edad. La ALU se presentó en la clase 14.
    d. Computadora sin **memoria RAM:**
        Una computadora sin memoria RAM no podría almacenar ni acceder a la información de las variables, no podríamos recurrir a información en tiempo de ejecución como la que almacenamos en edad. La memoria RAM y su funcionamiento fueron vistos en las clases 10 y 11, mientras que las variables se empezaron a usar a partir de la clase 6.
    e. Computadora sin **memoria *caché*:**

Una computadora sin memoria *caché* podría ejecutar el programa, ya que su presencia permite una mayor velocidad de acceso a la información pero no es imprescindible para el funcionamiento del programa presentado. Es posible que sin *caché* aumente el tiempo de ejecución. La memoria *caché* se vio en la clase 12.

f. Computadora sin **disco:**
   Una computadora sin disco no tendría lugar para almacenar el programa, lo que haría imposible su lectura y ejecución. También sería imposible modificarlo y guardar los cambios. El almacenamiento en disco fue abordado en la clase 11.
g. Computadora sin **compilador:**
   Una computadora sin compilador no podría convertir el programa a lenguaje máquina, lo que haría imposible su ejecución. La función de los compiladores fue vista en la clase 9.
h. Computadora sin **teclado:**
   Una computadora sin teclado nos imposibilitaría escribir el valor de la edad en tiempo de ejecución. La utilidad de los periféricos de entrada fue abordada en la clase 10.
i. Computadora sin **luz ni batería:**
   Una computadora sin luz ni batería no podría ejecutar ningún programa ya que requieren de electricidad para poder funcionar. Como se vio en la clase 14, incluso los 0 y los 1 que usualmente decimos que es lo único que "entiende" la computadora, es una abstracción de alto y bajo voltaje, es decir, una característica eléctrica.
j. Computadora sin **monitor o pantalla:**
   Una computadora sin monitor o pantalla nos impediría ver el cartel que indica que ingresemos la edad ni la edad que se tendrá al año siguiente. Se mencionaron los periféricos de salida en la clase 10.

Clase 16 - Sistemas operativos I

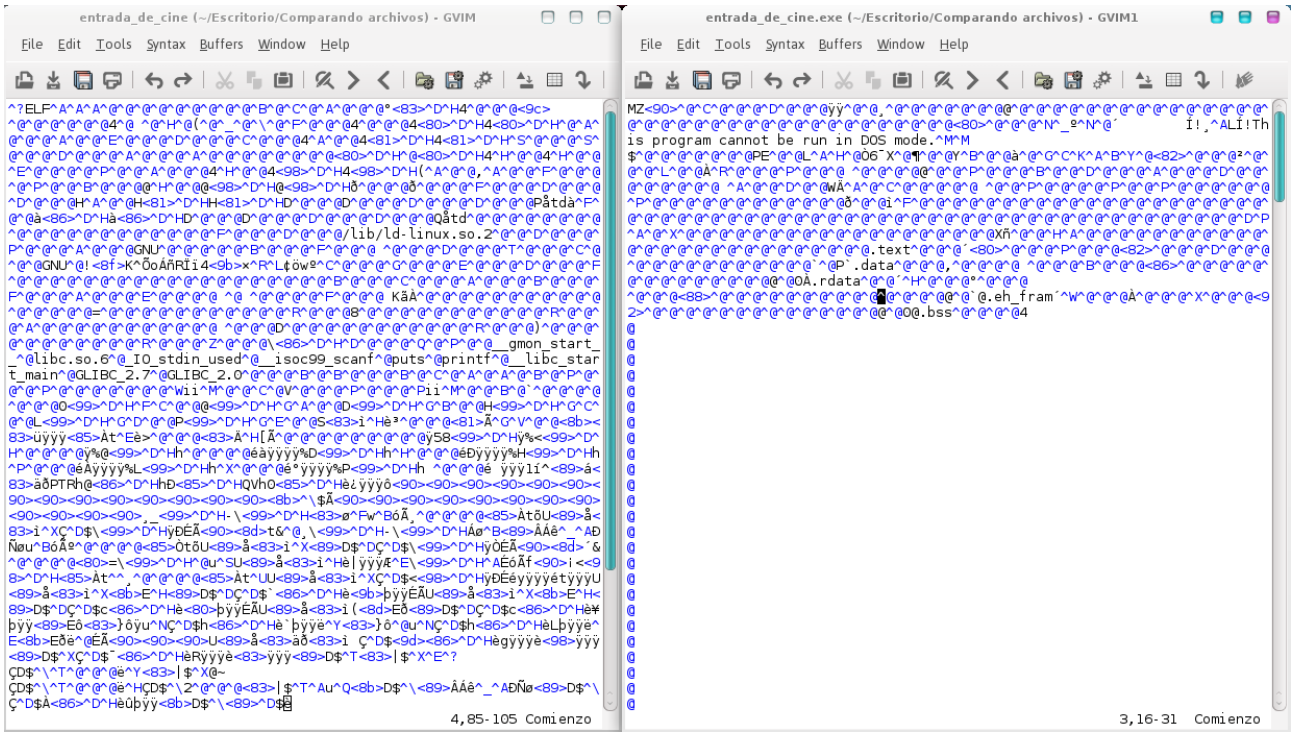
El objetivo de esta clase es hacer una introducción a los sistemas operativos partiendo de la forma en que se construyen los programas que creamos y que pueden ser ejecutados una computadora.

Al comienzo de la clase se retomará el programa creado en la clase 8 utilizando C. El/la docente preguntará a la clase si recuerdan qué es un archivo binario. Utilizando una computadora que cuente con dos sistemas operativos diferentes, explicará y mostrará a las y los estudiantes la compilación utilizando Windows (ver Anexo) para generar un archivo binario. Copiará el archivo `entrada_de_cine.exe` en alguna carpeta accesible desde el otro sistema operativo (en las computadoras de Conectar Igualdad se puede acceder al disco `F:` o `Datos` tanto desde Huayra como desde Windows). Una vez copiado este archivo, reiniciará la computadora en Huayra y nuevamente compilará el programa (tal como se indica en la clase 8) generando el archivo binario `entrada_de_cine`.

Una vez que se crearon los binarios se realizará un breve intercambio con las y los estudiantes:

- ¿El programa era exactamente igual?
- ¿Creen que los archivos binarios son iguales? ¿Cómo los compararían?

Para comparar los archivos, se puede observar el tamaño de cada uno. Se notará con facilidad que el binario generado en Windows es notablemente más pesado que el generado en Huayra. Luego, se abrirán ambos utilizando el editor GVim, obteniendo resultados similares al de la imagen, dejando en evidencia que no son exactamente iguales.



Luego de la comparación se plantearán los siguientes interrogantes:

- ¿Por qué creen que son distintos?
- Si intercambio los archivos binarios, ¿funcionarán en el otro sistema operativo?

Se espera que las y los estudiantes atribuyan las diferencias al uso de distintos sistemas operativos. Luego, se discutirá sobre la función de los sistemas operativos y por qué creen que generaron diferentes resultados.

A modo de síntesis, el/la docente expondrá brevemente que el sistema operativo es el encargado de mediar entre el hardware, es decir, todos los componentes físicos de la computadora, y el software que usamos. Por ejemplo, si para hacer un programa en C como el que se utilizó en clase se requiriera comprender en detalle el funcionamiento de todos los componentes de hardware sería muy difícil escribir ése o cualquier otro programa sencillo. El sistema operativo es el que le presenta a los programas una versión de la computadora más abstracta y más simple, además de ocuparse de administrar el funcionamiento de todos los componentes del hardware. Es decir, el sistema operativo es una pieza de software que hace de intermediario entre los programas del usuario y el hardware. La diferencia entre los dos archivos binarios se debe a que la forma en que cada sistema operativo hace esa mediación entre el programa y el hardware es diferente.

En este punto se pedirá a las y los estudiantes que mencionen distintos sistemas operativos[23] que conozcan (Huayra, Windows, GNU/Linux, Unix, Android, iOS, Windows Phone, Mac OS, Ubuntu, Debian, Windows Server, etc.).

Luego, el/la docente explicará que aquellas instrucciones vinculadas directamente con el funcionamiento del hardware son mediadas por el sistema operativo para evitar comportamientos inesperados o no deseados a través de lo que se conoce como *kernel* o núcleo del sistema. En este punto formulará a la clase las siguientes preguntas:

- ¿Qué pasa cuando se conecta un nuevo dispositivo a la computadora?
- ¿Todos los dispositivos son iguales? ¿Funcionan igual?
- ¿El sistema operativo reconoce automáticamente todas las versiones y variedades de dispositivos?
- ¿Cómo hace el sistema operativo para permitir el correcto uso de ese nuevo dispositivo?
- ¿Alguna vez instalaron una impresora, pantalla o conectaron un USB en su computadora? ¿Qué tuvieron que hacer?

A partir de las preguntas disparadoras, se presentará en clase el concepto de **driver** o **controlador** de dispositivos, un tipo de software que se ocupa de comunicarse con un dispositivo específico dándole comandos y aceptando respuestas. En general, los fabricantes del dispositivo

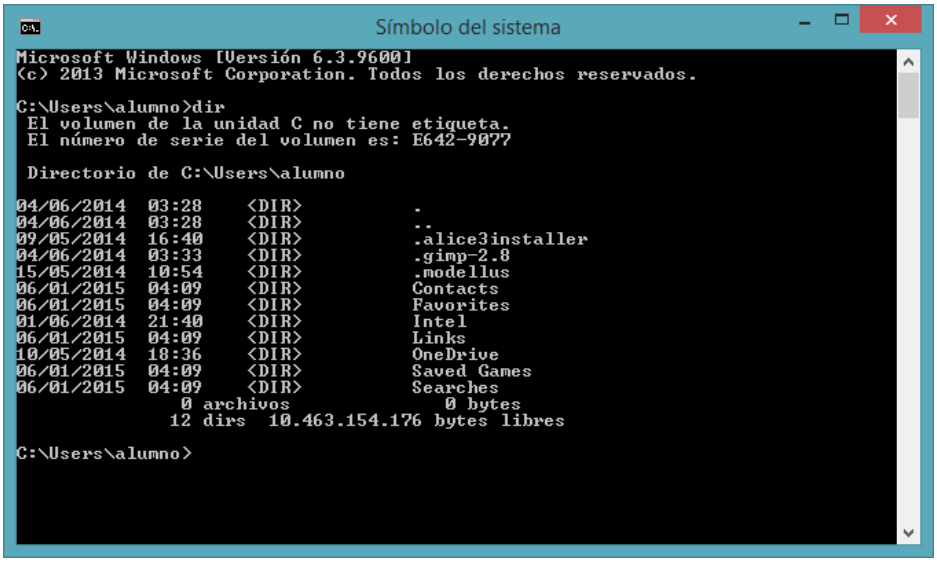
[23] En este curso no se diferenciarán los conceptos de sistema operativo y distribución de sistema operativo.

se encargan de desarrollar una versión de controlador adecuada para cada sistema operativo para que sea compatible.

El/la docente le propondrá a las y los estudiantes que decidan entre varias opciones cuáles piezas de software son parte del sistema operativo y cuáles no: juegos preinstalados en el celular y la PC, calculadora, administrador de tareas, planilla de cálculo, etc.

El/la docente explicará que así como los controladores o drivers facilitan la interacción entre el sistema operativo y los dispositivos, hay un software para que el usuario pueda interactuar con el sistema operativo y utilizar otros programas de manera más clara y sencilla: la **interfaz de usuario**. Buscará destacar que por más que algunos programas estén incluidos desde la instalación del sistema operativo no significa que sean parte del mismo.

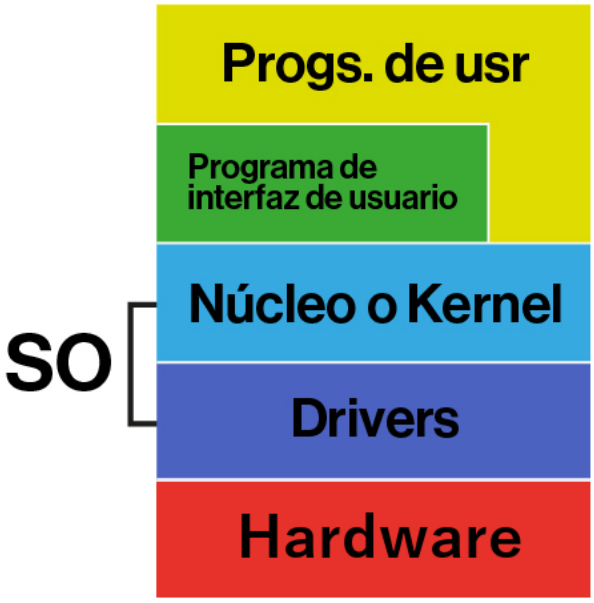
Las y los estudiantes indagarán en grupos sobre algunas posibilidades de las interfaces que poseen los sistemas operativos de sus celulares y computadoras, indicando similitudes y diferencias entre ellos, y listando algunas funciones de la interfaz de usuario que consideren importantes. El/la docente indicará que la interfaz puede ser gráfica o de texto (línea de comandos) y la mayoría de los celulares cuentan con interfaces táctiles. Se pueden mostrar algunos comandos para conocer una interfaz textual como son la Terminal de Huayra y el CMD de Windows (por ejemplo, listar los archivos de un directorio o carpeta: **ls** y **dir** respectivamente).



Algunas de las funcionalidades accesibles desde la interfaz que se espera que surjan son:

- Encendido y apagado de la computadora
- Manejo de archivos y directorios
- Administrar conexiones
- Configuración de la interfaz
- Información sobre el hardware
- Configuración de algunas funciones de la máquina

Para finalizar, entre toda la clase se completará un diagrama similar al siguiente, que permitirá visualizar en capas el hardware, los drivers o controladores de dispositivos, el Kernel o núcleo, la interfaz de usuario, y los programas o aplicaciones de usuario.



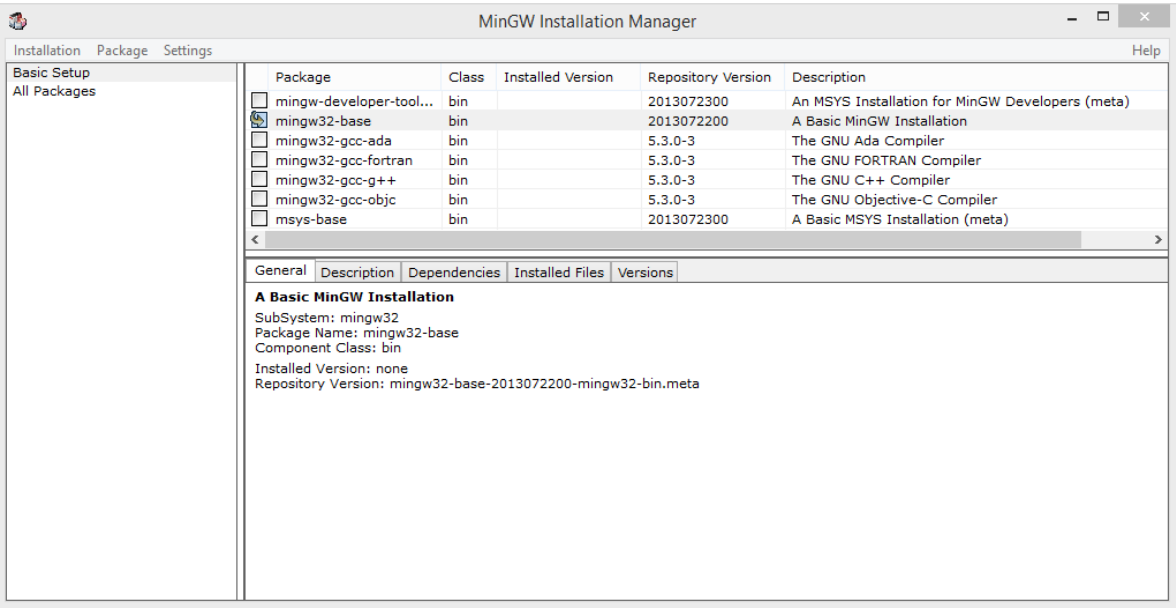
➢ *Anexo: cómo compilar el archivo entrada_de_cine.c en Windows*

Para poder compilar un archivo .c en Windows es necesario tener instalado algún compilador, como por ejemplo MinGW.

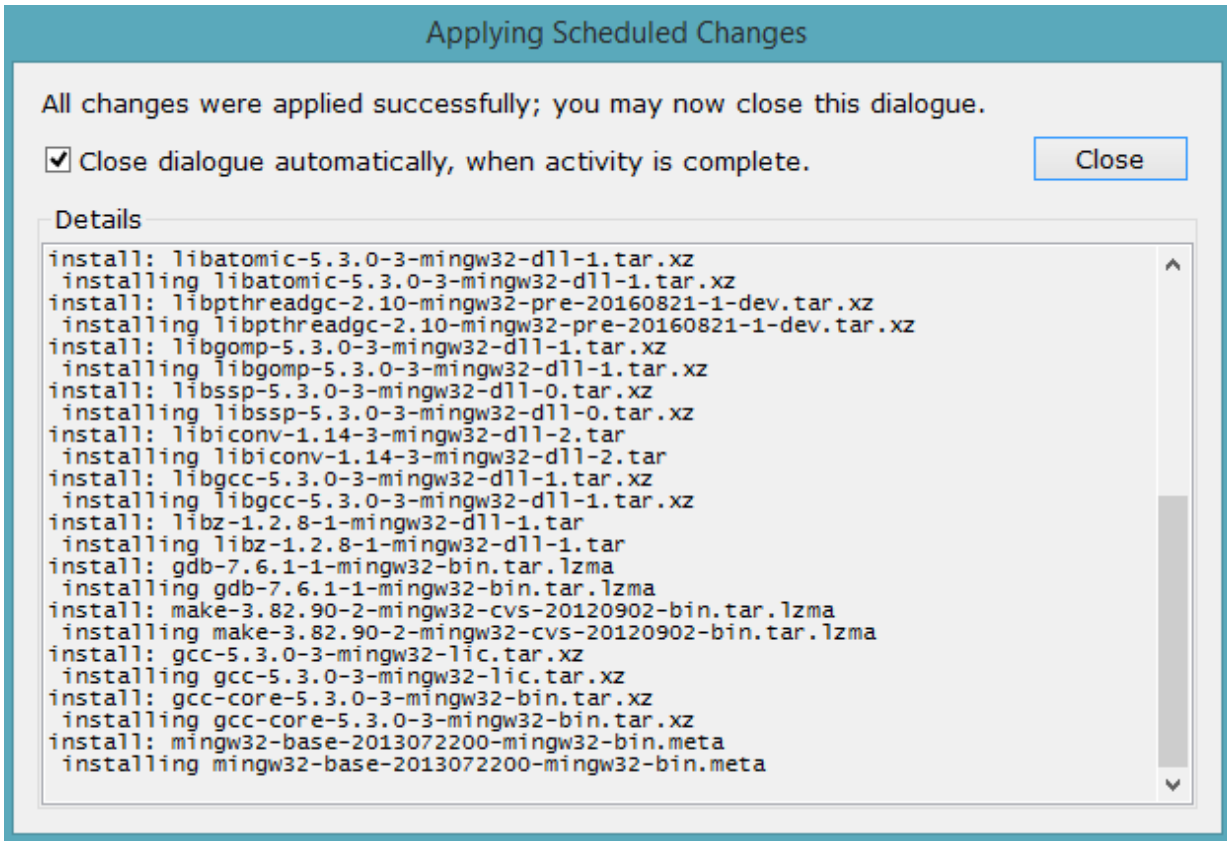
A continuación se describen brevemente los pasos a seguir para instalar y compilar con MinGW.

Instalación de MinGW

1. Descargar MinGW desde el sitio web http://www.mingw.org/ → *Downloads*
2. Se ejecutará el archivo descargado y se seguirán las siguientes instrucciones: *Install* → *Continue* → *Continue*
3. Se seleccionará *All Packages* en el menú de la izquierda y mingw32-base haciendo clic en el cuadrado junto al nombre. “Mark for Installation” marcará la opción seleccionada y otras complementarias que requiere para funcionar correctamente.



4. Luego en la barra de menú *Installation→Apply Changes→Apply*



5. Para finalizar la instalación se presionar *Close.*

6. Para poder compilar en Windows, es necesario modificar algunos aspectos de la configuración:

Panel de control → *Sistema y seguridad* → *Sistema*→ *Configuración avanzada del sistema* → *Opciones avanzadas* → *Variables de entorno*
Una vez dentro de variables de entorno, se agregará en “Variables de usuario” una nueva opción presionando: *Nueva…*
Nombre de la variable: `PATH`
Valor de la variable: `C:\MinGW\bin`

Al concluir la instalación y configuración de MinGW estaremos en condiciones de compilar el archivo entrada_de_cine.c

Compilación en Windows

Abrir una terminal de windows desde *Inicio* → *Sistema de Windows* → *Símbolo del sistema* (o escribiendo CMD en el buscador de aplicaciones) y ubicándose en la carpeta donde tenemos el archivo `entrada_de_cine.c` escribir:

```
gcc -o entrada_de_cine funciones.c entrada_de_cine.c
```

Al presionar *Enter* se generará el archivo entrada_de_cine.exe que es el binario necesario para el desarrollo de la clase.

Clase 17 - Sistemas operativos II

En esta clase se continuarán identificando las partes y funciones de los sistemas operativos. Para comenzar, el/la docente invitará al curso a responder algunas preguntas:

- ¿Qué está haciendo su celular en este momento?
- ¿Puede sonar el teléfono mientras sacan una foto?
- ¿Quién se ocupa de regular el funcionamiento de todas esas tareas?

Se espera que las y los estudiantes identifiquen que en sus dispositivos pueden estar ocurriendo cosas que no dependen directamente de las aplicaciones que están utilizando ellos en ese momento y que se analice de manera similar el comportamiento de una computadora. Luego, explicará que otra de las funciones del sistema operativo es administrar la ejecución de los distintos procesos. Un **proceso**, a grandes rasgos, es un programa que se está ejecutando. Por ejemplo, al ejecutar el programa "`entrada_de_cine`" se genera un proceso. Los procesos tienen asociado un espacio de la memoria RAM determinado donde poder leer y escribir, entre otros recursos del sistema. El sistema operativo debe decidir cuándo dejar de ejecutar un proceso y pasar a otro. Es importante remarcar que la diferencia entre un programa y un proceso es que el programa puede ejecutarse una, varias o ninguna vez y un proceso existe o se crea en el momento de ejecutar dicho programa.

Las y los estudiantes analizarán en grupos el Administrador de Windows y el Monitor del sistema de Huayra y observarán los procesos que corren en sus computadoras y compararán el rendimiento y el uso de recursos de algunas aplicaciones:

- Alice
- Calculadora
- Procesador de texto

Además, se les indicará cómo mirar los procesos en ejecución en celulares con Android:

Configuración → Acerca del teléfono

Una vez aquí, deberán tocar varias veces en *Número de compilación* hasta que indique que se cuenta con los permisos de programador. Luego de esto, ya estará habilitado el acceso a un nuevo menú llamado "Programador" que nos permite acceder a información y opciones pensadas especialmente para aquellos que crean aplicaciones de celulares.

Ingresando a:

se verán los procesos que están corriendo en el celular y la memoria RAM que utilizan. Se pedirá a las y los estudiantes que identifiquen y registren las aplicaciones que están en ejecución en sus teléfonos y el consumo de recursos que indica el dispositivo.

Cuando una aplicación no responde, es posible finalizar los proceso en ejecución sin tener que reiniciar el sistema por no poder utilizarlo. En Huayra y otras distribuciones de Linux se lo puede hacer escribiendo en la Terminal *xkill* y haciendo clic en la ventana correspondiente a la aplicación que “se colgó”. En Windows, mediante el administrador de Windows, se puede finalizar un proceso en ejecución.

Es importante enfatizar respecto a no cerrar los procesos en ejecución pertenecientes al sistema, tanto en la computadora como en los celulares, dando lugar a las siguientes preguntas al respecto:

- ¿Por qué creen que tuvimos que seguir pasos poco usuales y “escondidos”, como tocar varias veces el “Número de compilación” para acceder a más opciones del celular?
- ¿Alguno de ustedes bloquea con pin o contraseña su celular? ¿Por qué? ¿Y la computadora?
- ¿Cuántas personas usan su celular? ¿Y su computadora?
- ¿Algunas vez leyeron los permisos de las aplicaciones instaladas en su celular? ¿Tiene sentido que una linterna pueda acceder al correo electrónico o los contactos?

Tomando como punto de partida las siguientes preguntas, se espera que las y los estudiantes identifiquen la importancia de resguardar el acceso a determinados archivos y funcionalidades. La protección no se aplica únicamente a las y los usuarios. Se debe poder garantizar que los distintos tipos de memoria, los archivos y otros recursos solamente sean accedidos por aquellos procesos debidamente autorizados por el sistema operativo. En la clase 20 se profundizará sobre seguridad informática y software malicioso, pero pueden mencionarse ejemplos al respecto.

A continuación, se pedirá a las y los estudiantes que indaguen sobre los usuarios de sus computadoras y respondan:

- ¿Tienen más de un usuario?
- ¿Cuáles son?
- ¿Creen que pueden instalar cualquier cosa con ellos?
- ¿Tienen contraseña?
- ¿Qué opciones encuentran para los tipos de usuarios?

El/la docente invitará a la clase a que imaginen, por ejemplo, que quieren prestarle la compu a un hermano menor pero no quieren que modifique los trabajos prácticos de Tecnologías ni les cambie la configuración del escritorio. La propuesta de actividad será crear un nuevo usuario con permisos limitados para el hermano menor.

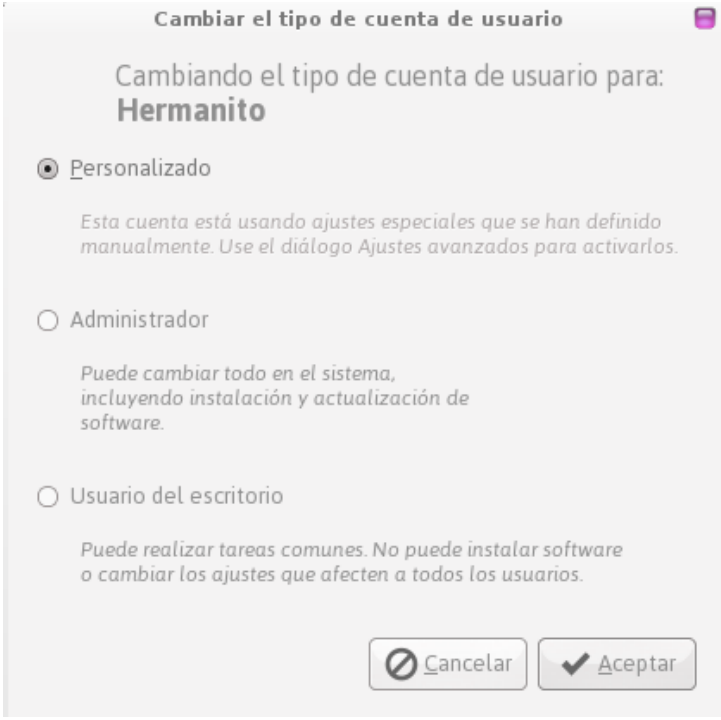
El/la docente aprovechará para contar que en Huayra todas las computadoras se entregan con el mismo usuario (alumno) y clave (alumno), es decir que cualquiera puede acceder a usuarios con permisos de administrador si no la modifican. Por eso, se les sugerirá, en caso de que no lo hubieran hecho, cambiar nombre y contraseña.

¿Qué es un usuario Administrador? Un usuario que tiene mayores privilegios de uso y configuración. Por ejemplo, puede crear y borrar nuevos usuarios e instalar distintos tipos de software. Se espera que las y los estudiantes identifiquen que la cuenta para el hermano menor no debería tener acceso a estas modificaciones.

- **Crear un nuevo usuario en Huayra**
  Sistema → Centro de control → Usuarios y grupos → Añadir

Ajustes de los usuarios
Alumno
alumno
Alumno
Cambiar...
Tipo de cuenta: Personalizado
Cambiar...
Contraseña: Preguntar al iniciar sesión
Cambiar...
Añadir
Eliminar
Gestionar grupos
Ajustes avanzados
Ayuda
Cerrar

Crear un usuario nuevo
Crear un usuario nuevo
Nombre:
Usuario:
El nombre de usuario debe consistir de:
letras en minúscula del alfabeto inglés
dígitos
cualquiera de los caracteres «.», «,» y «_»
Cancelar
Aceptar


Cambiar el tipo de cuenta de usuario
Cambiando el tipo de cuenta de usuario para:
Hermanito
Personalizado
Esta cuenta está usando ajustes especiales que se han definido manualmente. Use el diálogo Ajustes avanzados para activarlos.
Administrador
Puede cambiar todo en el sistema, incluyendo instalación y actualización de software.
Usuario del escritorio
Puede realizar tareas comunes. No puede instalar software o cambiar los ajustes que afecten a todos los usuarios.
Cancelar
Aceptar

- **Crear un nuevo usuario en Windows**
  Configuración de PC → Cuentas → Otras cuentas → Agregar cuenta

- **Para modificar las cuentas existentes en Windows**
  Panel de control → Cuentas de usuario → Cambiar el tipo de cuenta

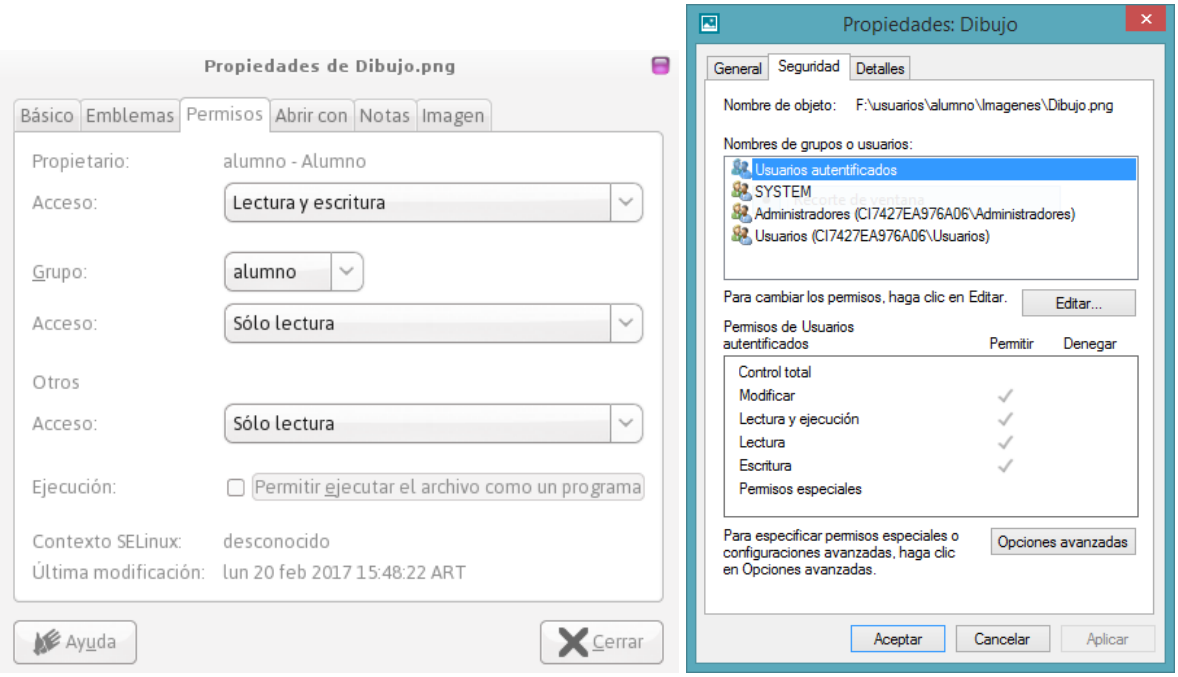
Una vez que los usuarios dispongan de sus cuentas y claves de Administrador, podrán indagar respecto a los permisos de cada archivo.

Los archivos pueden compartirse entre usuarios o grupos de usuarios, pero se puede elegir con cuánta libertad para leerlos, modificarlos (escribirlos) o ejecutarlos.

Para modificar o visualizar los permisos se deberá hacer clic derecho en el archivo o directorio, seleccionar *Propiedades* e ir a la pestaña de *Seguridad* en Windows o *Permisos* en Huayra.
Se espera que las y los estudiantes comprendan los tipos de permisos:

- **Ejecución**: en el caso de los archivos podrán ejecutarse y en el de los directorios acceder y ejecutar su contenido (siempre y cuando también lo permita)
- **Lectura**: se podrá acceder y leer.
- **Escritura**: se podrá escribir y modificar.

Las y los estudiantes crearán un archivo de imagen (con Paint, MyPaint o similar) usando su usuario Administrador, le pondrán permisos de “Sólo lectura” para otros usuarios e intentarán modificarlo y guardarlo desde el nuevo usuario creado con menos permisos. Luego intentarán modificar los permisos. Finalmente, harán lo mismo pero creando la imagen desde el usuario de menores privilegios para poder observar que luego el Administrador podrá modificarlos.

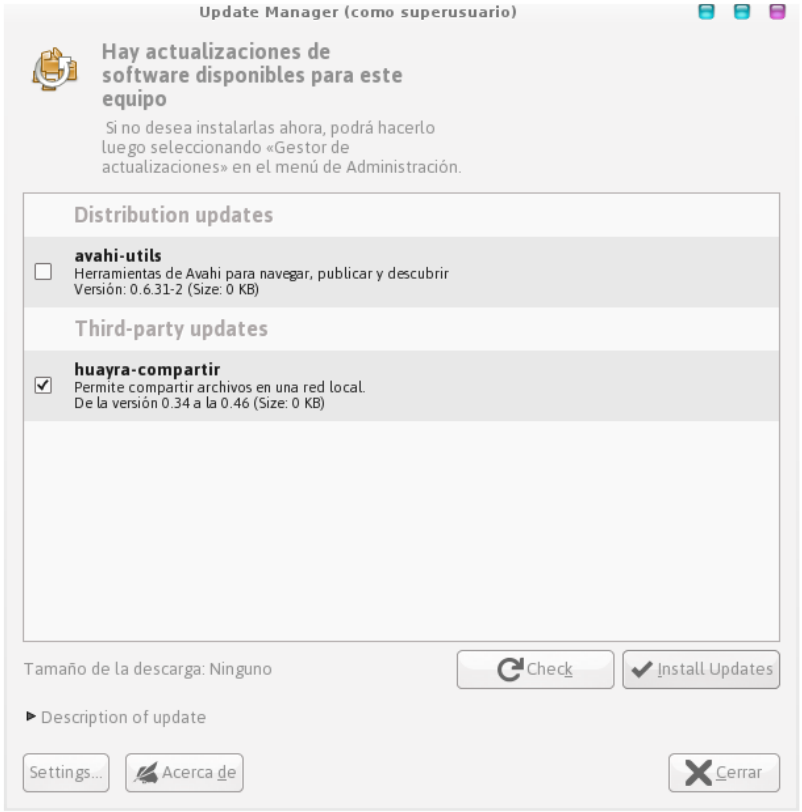


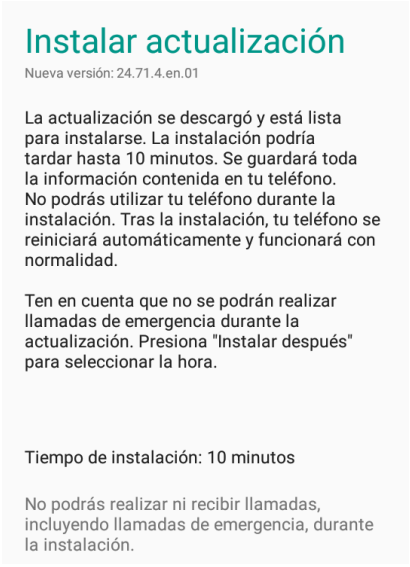
Clase 18 - Sistemas operativos III

La clase comenzará con algunas preguntas:

- ¿Actualizaron alguna aplicación del celular? ¿Por qué?
- ¿Tuvieron que actualizar Android, Windows o Huayra?
- ¿Por qué creen que es importante actualizar sus aplicaciones? ¿Y el sistema operativo?
- ¿De dónde sacan las aplicaciones, juegos y otros programas que instalan?

El/la docente tomará como punto de partida estas preguntas para explicar que una vez lanzados a la calle, los sistemas operativos siguen mejorando características y corrigiendo errores. Además, el uso masivo permite detectar fallas de seguridad o errores (*bugs*) que en el proceso de desarrollo no necesariamente fueron halladas. La instalación de un nuevo sistema operativo demanda tiempo e implica modificaciones en varias funcionalidades, por eso se realizan actualizaciones periódicas para corregir o modificar algunas partes y no se lanza uno nuevo completo.





Las y los estudiantes seguramente mencionarán que algunos celulares cuentan con “Tiendas” para descargar aplicaciones pagas o gratuitas como “Play Store” en Android. Windows también incluye en algunas distribuciones una Tienda para adquirir software. Se aprovechará para contar que Huayra y otras distribuciones de Linux cuentan con centros de software y sistemas de manejo de paquetes. Los paquetes se denominan así porque incluyen varios programas que se distribuyen “empaquetados” todos juntos. Su instalación, actualización y desinstalación se realiza a través de un gestor de paquetes como, por ejemplo, Synaptic. Estos programas sirven para mantener estable el sistema al momento de instalar o desinstalar un programa o una actualización ya que se ocupan de verificar que cualquier programa que se instale cuente con aquellos otros programas necesarios para su correcto funcionamiento. También chequean en Internet si hay actualizaciones disponibles para nuestro sistema operativo. Los gestores requieren la clave de Administrador para funcionar. Los programas para instalar en Windows pueden descargarse con facilidad ya que su uso está bastante extendido pero la descarga desde algunos sitios no siempre es segura (este tema se profundizará en la clase 20).

Luego de esta exposición, el/la docente propiciará una discusión respecto a cuáles son los criterios que pueden utilizar para elegir qué sistema operativo instalar en su computadora y abordar aspectos que aún no se hayan profundizado en clase. En función de las respuestas, el/la docente sumará algunas preguntas y nuevos criterios hasta considerar varios de ellos:

- ¿Existe una versión de los programas que quiero usar para cualquier sistema operativo?
  Existen programas que no cuentan con versiones compatibles con algunos sistemas operativos.
- ¿Tengo que utilizar sí o sí un programa determinado o puedo utilizar uno diferente con funciones similares?
  Si se requiere usar un software puntual para un trabajo o actividad escolar, tiene sentido buscar un sistema operativo compatible. Pero muchas veces encontramos programas diferentes con las mismas funcionalidades. Por ejemplo, no existen grandes diferencias entre usar la calculadora de Huayra, Windows o el celular para realizar una suma. No tendría sentido condicionar la elección por una herramienta que cuenta con versiones equivalentes en cualquiera sistema.
- ¿Tengo limitaciones de recursos? ¿Mi computadora es “vieja”?
  Algunos sistemas operativos son especialmente compatibles con computadoras de pocos recursos o más viejas. Por ejemplo Lubuntu y LXLE son opciones ideales en estos casos.
- ¿Necesito usar hardware específico?
  Si queremos conectar algún dispositivo especial a nuestra computadora, debemos investigar si existen drivers o controladores para usarlo con nuestro sistema operativo.
- ¿Existen diferencias de seguridad?
  En general, Windows se encuentra muchísimo más expuesto a virus y software malicioso. Otras opciones como Huayra o Mac OS suelen ser más seguros en este sentido. En la clase 20 se profundizará sobre este aspecto.

- ¿Cuál es el precio de cada sistema operativo?
  Existen sistemas operativos que pueden obtenerse gratuitamente y otros que se comercializan.
- ¿Quiero usar la computadora para programar?
  En general Linux presenta gran variedad de recursos para programar en diversos lenguajes, pero si queremos usar alguno muy específico y vinculado a otro sistema, puede convenirnos otra opción. Por ejemplo, para hacer aplicaciones para un celular iPhone puede resultarnos más práctico utilizar una computadora con Mac OS.

Para concluir, el/la docente mencionará que es posible instalar más de un sistema operativo en una misma computadora. Por ejemplo, las netbooks de Conectar Igualdad cuentan con dos: Huayra y Windows.

Como actividad, el/la docente retomará el trabajo práctico de la clase 15 y pedirá a las y los estudiantes que escriban y entreguen qué pasaría si quisieran ejecutar el programa de la edad y su computadora no tuviera sistema operativo.

Se espera que en la respuesta se identifique la imposibilidad de usar una computadora tal y como se acostumbra. La falta de una interfaz limita al usuario tanto para ejecutar un programa como para recurrir a la mayoría de los componentes de hardware. Tampoco se contará con drivers o controladores para utilizar los distintos componentes de la manera esperada ni un software que administre el comportamiento armónico de toda la computadora.

Clase 19 - Software libre

El objetivo de esta clase es que las y los estudiantes comprendan la diferencia entre **software propietario** (también denominado *privativo*) y **software libre** (considerado -aunque con diferencias- software de código *fuente* abierto). La clase permitirá repasar contenidos anteriores como la diferencia entre el código fuente y el código binario de un programa.

Para que las y los estudiantes reconozcan de qué manera sus prácticas cotidianas con las computadoras están atravesadas por la problemática de la propiedad del software y las libertades de sus usuarios, se comenzará la clase con una serie de preguntas que guiarán un debate colectivo.

Se les pedirá a las y los estudiantes que reflexionen sobre el modo en que consiguieron el último libro o canción favoritos.

- ¿En qué formato los tienen: digital o analógico?
- ¿Cómo los consiguieron? ¿Fue gratis o pagaron por ellos?
- ¿Lo han copiado y compartido con otras amigas o amigos?

Luego de unos minutos de debate se les pedirá que respondan las mismas preguntas en relación al último videojuego que jugaron y al último programa que descargaron.

- ¿Cómo consiguieron estos programas? ¿Fueron gratis o pagaron por ellos?
- ¿Lo han copiado y compartido con otras amigas o amigos?

Justamente en esta clase nos preguntaremos: ¿qué derechos y libertades tenemos cuando usamos los programas más cotidianos como el sistema operativo, los reproductores de música, procesadores de texto, videojuegos o editores de imágenes?

Los programas de computadoras, así como las canciones, novelas y pinturas son obras creativas del intelecto humano reguladas por leyes de propiedad intelectual. Es decir, cuando plasmamos en una obra una idea que hasta recién vivía en nuestra cabeza (bajo la forma de líneas de código, palabras o notas musicales), de manera inmediata como autoras y autores comenzamos a tener derechos sobre esa obra.

Estos derechos son de dos tipos: los derechos de autor y los derechos de copia (o copyright). El derecho de autor le asegura a las autoras y autores el reconocimiento como creadores; y gracias al derecho de copia serán ellos únicamente quienes permitan publicar, copiar, modificar y difundir su obra. En la mayoría de los casos los autores venden sus derechos de copia a intermediarios como editoriales o discográficas, quienes se ocupan de la publicación y difusión de la obra.

Pero ¿qué significa que los programas de computadora que usamos o estudiamos se vean regulados por esas leyes estos derechos?

Por más que hayamos comprado una novela o un programa informático necesitamos el permiso del autor o de la editorial o discográfica para copiar esa obra (fotocopiarla, descargarla o imprimirla), para modificarla o mejorarla según nuestras necesidades, para traducirla y distribuir copias ya sea regalándolas o vendiéndolas.

Para avanzar con la distinción entre software libre y software propietario se les presentará a las y los estudiantes el siguiente cuadro que categoriza diferentes programas según si son libres o propietarios:

	Software Propietario	**Software Libre**
Sistema operativo	Windows | Mac OS X	GNU/Linux (Huayra, Ubuntu, Debian)
Ofimática	Microsoft Office	LibreOffice | OpenOffice
Navegador	Chrome | Safari	Mozilla Firefox

Reproductor de videos	Reproductor Windows Media	VLC Media Player
Reproductor de audio	iTunes | Winamp	VLC Media Player
Editor de imágenes	Photoshop | Illustrator	Gimp | Inkscape
Editor de videos	Adobe Premiere	Avidemux

	Candy Crush, Age of Empires, Counter Strike, League of Legends	SuperTux
Videojuegos		

Cuadro 19.1: programas propietarios y programas libres

Para finalizar esta breve introducción se les pedirá a las y los estudiantes que indiquen programas de uso cotidiano en la computadora que no se encuentran presentes en el Cuadro 19.1[24].

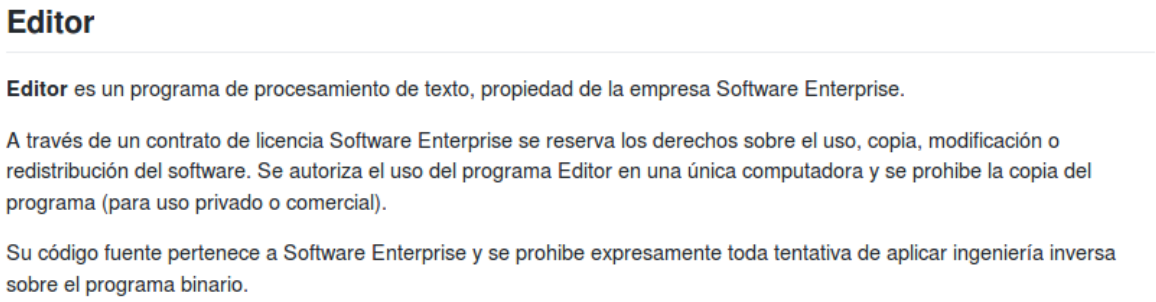
Programas libres vs Programas propietarios

A continuación se trabajará la diferencia entre un programa libre y un programa propietario.
Se comenzará resaltando que ambos programas no se distinguen por su funcionamiento sino que podrían funcionar de manera idéntica, dejándonos hacer las mismas cosas. En cambio, la diferencia entre un programa libre y uno propietario radica en las libertades que cada programa le otorga a sus usuarias y usuarios.

Para abordar este tema se trabajarán brevemente entre todas y todos dos situaciones cotidianas, para evaluar las libertades con las que las usuarias y usuarios cuentan al utilizar un programa.

Caso 1: Necesitamos editar un texto y para eso adquirimos el programa Editor, un programa propietario cuya dueña es una empresa de software. Al adquirir el programa nos enviaron un archivo binario para instalar el programa y aceptamos un contrato de licencia para su uso. En el siguiente apartado es posible leer un extracto del mismo:

[24] El tipo de licencia de un gran número de programas puede consultarse en https://prism-break.org/es/categories/windows/



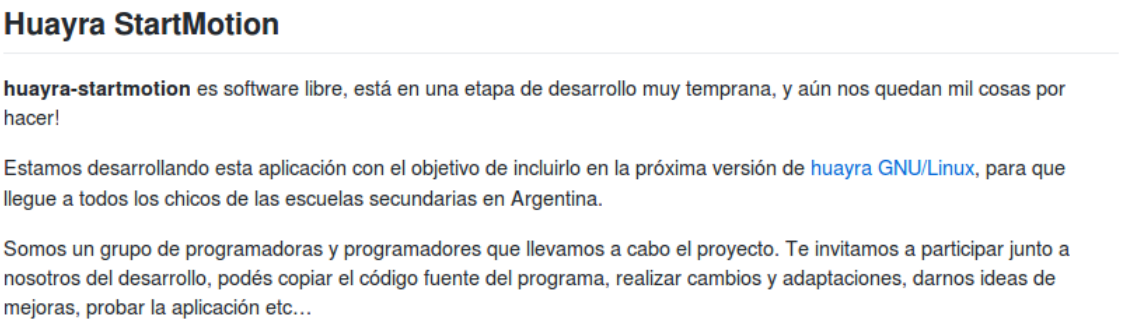
Editor

Editor es un programa de procesamiento de texto, propiedad de la empresa Software Enterprise.

A través de un contrato de licencia Software Enterprise se reserva los derechos sobre el uso, copia, modificación o redistribución del software. Se autoriza el uso del programa Editor en una única computadora y se prohibe la copia del programa (para uso privado o comercial).

Su código fuente pertenece a Software Enterprise y se prohibe expresamente toda tentativa de aplicar ingeniería inversa sobre el programa binario.

Caso 2: Necesitamos realizar un stop motion, para eso descargamos el programa Huayra Startmotion[25]. Es un programa con una licencia considerada libre y al descargarlo obtenemos tanto su código fuente como su archivo binario. En la página del proyecto se indica lo siguiente:



Huayra StartMotion

huayra-startmotion es software libre, está en una etapa de desarrollo muy temprana, y aún nos quedan mil cosas por hacer!

Estamos desarrollando esta aplicación con el objetivo de incluirlo en la próxima versión de huayra GNU/Linux, para que llegue a todos los chicos de las escuelas secundarias en Argentina.

Somos un grupo de programadoras y programadores que llevamos a cabo el proyecto. Te invitamos a participar junto a nosotros del desarrollo, podés copiar el código fuente del programa, realizar cambios y adaptaciones, darnos ideas de mejoras, probar la aplicación etc...

Se le pedirá a la clase que evalúe brevemente cuáles de las siguientes libertades tenemos al adquirir cada uno de los programas:

¿Cuáles son las "libertades" de las y los usuarios del programa?

- ¿Tenemos permiso para instalar el programa en otras máquinas?
- ¿Además del archivo binario podemos ver el código fuente del programa?
- ¿Tenemos permiso para darle una copia a nuestras amigas o amigos?
- Si nos damos cuenta que el programa no hace exactamente lo que queremos: ¿tenemos permiso para modificar el programa, mejorarlo o adaptarlo a nuestras necesidades?

A la hora de guiar el debate de los casos y considerar las respuestas de las y los estudiantes tener en cuenta el siguiente cuadro[26]:

Software Propietario	Software Libre
Las respuestas a las preguntas es **negativa**.	Las respuestas a las preguntas es **positiva**.

[25] Con fines educativos se toma un programa inventado basado en Huayra Stopmotion: https://github.com/HuayraLinux/huayra-stopmotion.
[26] Este esquema caracteriza en términos generales los programas propietarios y libres. Existen numerosas licencias de programas tanto libres como propietarios que se salen de este esquema.

Accedemos al archivo binario del programa.	Accedemos al archivo binario y al código fuente del programa.
Nos permite solamente instalar, usar y ejecutar una copia del programa como resultado de haber comprado una única licencia del mismo.	Nos permite instalar y ejecutar el programa, estudiar su código fuente, copiarlo e instalarlo en otras computadoras, mejorar el programa (editando su código fuente) y distribuir nuevas copias.

Como cierre del debate entonces se tendrán las herramientas para distinguir en términos generales un programa libre de uno propietario.

Como breve resumen se indicará que los **programas propietarios** son denominados **privativos** debido a que al usarlos debemos renunciar a nuestro derecho a estudiarlos, modificarlos y compartirlos, razón por la cual tampoco se nos da acceso al código fuente del programa.
En cambio un programa sólo será libre si le otorga a sus usuarios justamente la libertad de usarlo, estudiar su código fuente, mejorarlo y distribuir copias. Es por ello que si se da una respuesta afirmativa a todas las preguntas anteriores el programa será un programa libre.

Este derecho que tenemos como usuarias y usuarios de un programa libre se resume en las 4 libertades del software libre:

0) usar el programa con cualquier fin.
1) estudiar su funcionamiento y adaptarlo a nuestras necesidades.
2) distribuir copias (del modo que se nos ocurra).
3) mejorar el programa y distribuir a su vez dichas mejoras.

Propuesta: monografía

Por último, se les pedirá a las y los estudiantes que realicen una monografía de tarea de manera individual. Se trabajará en torno a dos casos, indicados a continuación, con la idea de que reflexionen en torno a la siguiente pregunta:

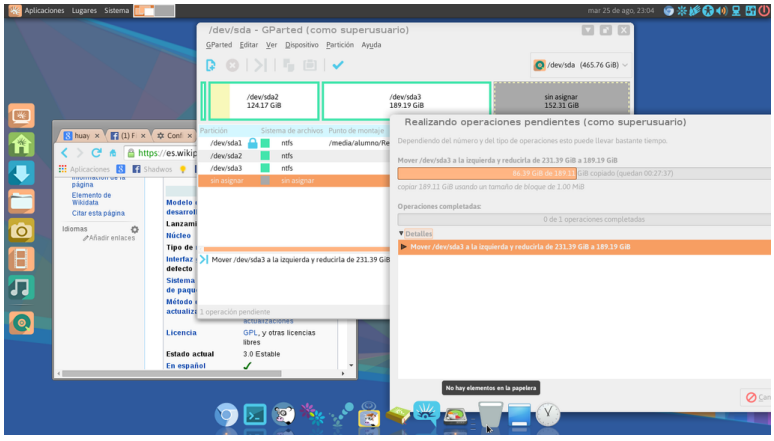
- ¿Para qué querríamos como usuarios de un programa conocer el código fuente de ese programa?

Se le asignará a cada estudiante uno de los dos ejemplos que se indican a continuación y que les servirá para guiar su razonamiento argumentativo.

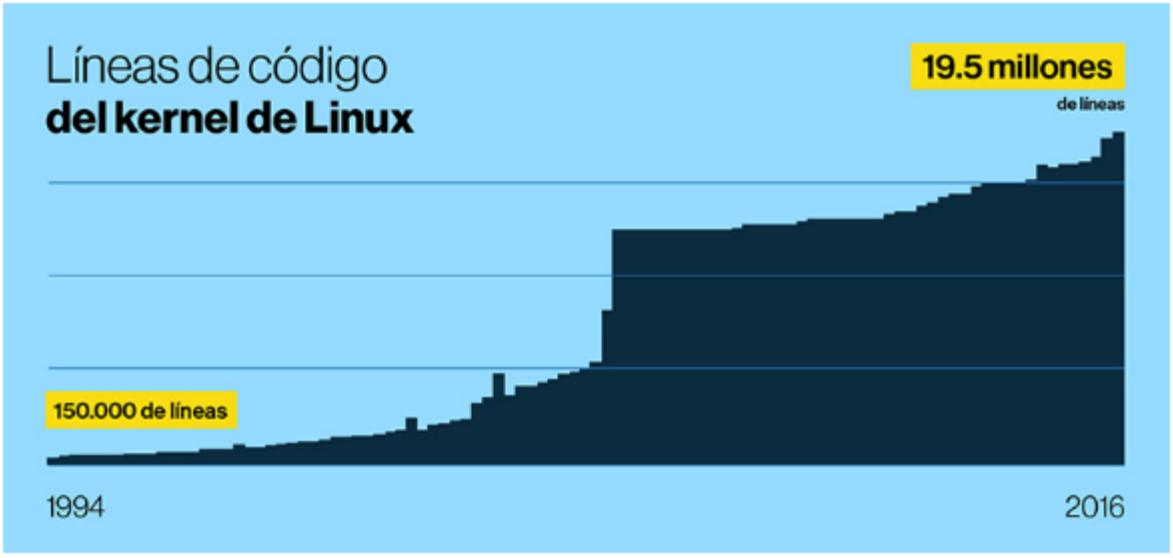
Los ejemplos propuestos son: Huayra, la distribución del sistema operativo GNU/Linux presente en las computadoras Conectar Igualdad, y un modelo de la muñeca Barbie de Estados Unidos que incluye como novedad un programa de reconocimiento y grabación de voz.

Caso I:

El equipo de desarrolladores de Huayra, basándose en el sistema operativo Debian, lo adaptaron a las necesidades de las escuelas de Argentina. En total, Debian cuenta con 419 millones de líneas de código de las cuales aproximadamente 19,5 millones corresponden al kernel de Linux[27]. En la actualidad existen casi 300 distribuciones del sistema operativo GNU/Linux basadas en el núcleo de Linux como Huayra.



Escritorio de Huayra, distribución GNU/Linux[28]



Evolución de la cantidad de líneas de código del kernel Linux

[27] Como dato de color, en 1994 el kernel tenía únicamente 150.000 líneas de código y se estima que costaría más de 3 mil millones de dólares contratar a un equipo de programadoras y programadores para desarrollarlo desde cero.
[28] Foto de Alejandro Avellaneda, trabajo propio, CC BY-SA 4.0

Costo de desarrollo del kernel Linux[29]

A partir de estos datos se reflexionará:

- ¿Hubiese sido posible crear Huayra sin haber contado con el kernel Linux, teniendo en cuenta el tiempo y dinero que habría implicado hacer un desarrollo propio completo de ese programa?
- Entonces: ¿para qué querríamos como usuarios de un programa tener acceso al código fuente de ese programa?

Caso II:

Como segundo ejemplo se tomará un modelo de la muñeca Barbie que cuenta en su interior con un micrófono y un programa de reconocimiento de voz. Se procederá a leer a las y los estudiantes un breve resumen del siguiente artículo que ilustra la polémica que causó dicha muñeca debido a denuncias que indican que no sólo graba las palabras de las niñas y niños (almacenándolas por dos años) y responde con frases prearmadas, sino que es acusada de espiarlos por filtrar detalles privados como la ubicación de las niñas y niños, no pudiendo asegurar que esa información caiga en manos de terceros.

A partir de estos datos se reflexionará:

- ¿Nos es posible saber con certeza qué está haciendo realmente la muñeca cuando activamos el programa de reconocimiento de voz?
- Entonces: ¿para qué querríamos como usuarios de un programa conocer el código fuente de ese programa?

[29] Fuente *https://www.suse.com/docrep/documents/sq1y0mk2uq/25_years_of_linux.pdf*

Como material para la y el docente, se espera que las redacciones tomen en cuenta que disponer del código fuente de un programa permite su modificación y reutilización, como fue el caso de Huayra con Debian, evitando el trabajo de tener que realizar y repetir todos su desarrollo desde cero. Como analogía se podrá pensar: ¿cuánto demoraríamos en escribir nosotros mismos el contenido de los 37 millones de artículos de wikipedia?

A su vez, en el caso de la muñeca resulta de importancia contar con el código fuente de un programa como condición para conocer qué hacen exactamente los programas que usamos diariamente. En el caso de la muñeca Barbie, un grupo de informáticos trabajó sobre los archivos binarios del programa de reconocimiento de voz de la muñeca logrando acceder al código fuente del programa[30] y descubriendo funcionalidades no deseadas dentro del mismo.

Se espera entonces que surja de las monografías la idea de que al no contar con el código fuente de un programa propietario no es posible controlar qué hace el programa -al menos no sin grandes dificultades.

Contenido complementario

Se proponen los siguientes contenidos complementarios si se dispone de tiempo adicional en la clase.

Gratis vs Libre

No necesariamente un programa privativo será pago ni un programa libre será gratuito. Se les preguntará a las y los estudiantes qué videojuegos o aplicaciones conocen de descarga gratuita pero -podrán verificarlo- no nos permiten estudiar cómo funcionan a partir de su código fuente. También es posible pagar por una copia de un programa libre, por ejemplo porque queremos una copia en CD o en un pendrive, o porque queremos contribuir económicamente con el proyecto. Sin embargo, como usuarios de un programa libre tendremos la libertad de estudiar su código fuente, copiarlo, modificarlo e incluso vender estas copias mejoradas (siempre y cuando quienes las compran cuenten también con las mismas libertades que nosotros al comprarlas). Igualmente vale destacar que la mayoría de los programas libres son también gratuitos.

Copyleft

¿Cómo es posible garantizar las libertades que otorga un programa libre? Así como los programas privativos tienen licencias propietarias que regulan su uso por las y los usuarios, los programas libres cuentan con licencias libres que garantizan de manera legal las libertades mencionadas anteriormente. Pero para que a medida que las copias modificadas y mejoradas de los programas inicialmente libres mantengan las libertades para sus usuarios, se creó el concepto de copyleft invirtiendo el término “copyright”[31]. Cuando un programa tiene una licencia que además de libre es “copyleft”, decimos que es un programa libre que le otorga a sus usuarios el derecho a ejecutarlo, copiarlo, modificarlo y distribuirlo. Más aún: todo aquel que reciba una copia de ese programa

[30] Se les puede recordar lo que hicieron en la clase 9 con BARF al obtener el código de máquina a partir de un archivo binario. En el caso de la Barbie se utilizaron técnicas similares.
[31] Dado que en inglés el derecho de autor, o más específicamente el derecho de copia se indica bajo el término copyright. El nombre “copyleft” indica que el programa no cuenta con derecho de copia, sino que literalmente “deja copiar” o “libres de copia”. Un ejemplo de una licencia de tipo copyleft es la licencia “Pública General de GNU” (en inglés “GNU General Public License”).

deberá tener garantizadas las mismas libertades para copiarlo, mejorarlo y distribuirlo a su vez. La condición será no sustraer esas capacidades a las y los usuarios de los programas mejorados. Es decir, ningún programador podrá modificar un programa libre licenciado bajo copyleft y luego prohibir a algún usuario realizar una copia de su programa mejorado. De esta manera los programas libres (su código fuente y su archivo binario) quedan siempre a disposición de la comunidad que se encarga de mejorar su funcionamiento y de controlar y auditar aquello que hacen los programas.

Clase 20 - Software malicioso

Se comenzará la clase con una breve discusión alrededor de los conceptos de virus informático, hackers y hackeo, buscando que afloren las concepciones previas y las dudas sobre estos temas [32]. Como cierre de esta breve discusión se propondrá la categoría de *software malicioso*, es decir, aquel pensado para hacer daño u obtener una ventaja ilegítima, y se dedicará el resto de la clase a entenderlo en más detalle.

Como primera aproximación para comprender cómo funciona buena parte del software malicioso es necesario problematizar el concepto de *vulnerabilidad* en los programas de computadora. Como punto de entrada para trabajar el concepto, se demostrará a la clase el siguiente programa escrito en algún lenguaje de programación y se les pedirá que interpreten qué es lo que hace:

```
imprimir_un_texto("Ingrese su usuario: ");
usuario = leer_usuario();
imprimir_un_texto("Ingrese su contraseña: ");
contraseña = leer_contraseña();

if (existe(usuario)) {
      if (es_correcto(usuario, contraseña)) {
            imprimir_un_texto("¡Te damos la bienvenida a tu cuenta!");
      } else {
            imprimir_un_texto("El usuario o contraseña es incorrecto.");
      }
      ingresar_a_la_cuenta(usuario);
}
```

El programa anterior debería verificar si al ingresar un usuario y una contraseña, dicho usuario existe en el sistema y la contraseña para ese usuario es la correcta. Sin embargo, independientemente de que la contraseña sea o no correcta, luego del **`if {} else {}`** se permite ingresar al usuario. Esto implica que si se ingresa un usuario válido, cualquier contraseña funcionará para entrar al sistema. ¿Cómo se podría subsanar de manera sencilla este problema? Las y los estudiantes deberían poder observar que habría que mover la línea que permite ingresar al usuario dentro del segundo **`if {}`**:

```
imprimir_un_texto("Ingrese su usuario: ");
usuario = leer_usuario();
imprimir_un_texto("Ingrese su contraseña: ");
contraseña = leer_contraseña();

if (existe(usuario)) {
      if (es_correcto(usuario, contraseña)) {
```

[32] En este marco será interesante traer sobre la mesa la diferencia entre la noción de criminal informático y hacker, entendiendo este último como una persona que explora los límites de la tecnología con el fin de mejorarla haciendo públicas vulnerabilidades y fallas de seguridad.

```
                imprimir_un_texto("¡Bienvenido a tu cuenta!");
                ingresar_a_la_cuenta(usuario);
        } else {
                imprimir_un_texto("El usuario o contraseña es incorrecto.");
        }
}
```

Mediante este ejemplo se puede ver que uno puede tener una idea correcta del programa que quiere hacer pero luego introducir un pequeño error en el código del programa, el cual termina teniendo o permite tener un comportamiento diferente al que esperábamos. A este tipo de errores, que afectan a la seguridad de un sistema, se los conoce como *vulnerabilidades* o *agujeros de seguridad*.

A modo de ejemplo se pueden contar algunos hallazgos recientes de importantes vulnerabilidades encontradas en sistemas muy utilizados. Una de las más resonantes de los últimos años es el caso del famoso Heartbleed que afectó a sitios populares como Amazon, Pinterest, SourceForge, Tumblr, Wikimedia, etc.

A continuación, se retomará la discusión inicial preguntando a la clase si ahora se imaginan qué es un virus de computadora y si conocen alguno. El objetivo es guiar la conversación para comprender que un virus es un programa de computadora que afecta la seguridad de un sistema o un equipo y que aprovecha una vulnerabilidad existente en ese sistema o equipo. Se trabajará también con las siguientes preguntas:

- ¿Las vulnerabilidades son siempre errores o pueden ser decisiones para que el sistema sea más fácil de usar?
- ¿Existen sistemas 100% seguros (pensar en casos no informáticos)?

Para reflexionar se puede preguntar también cuál es la motivación de quienes hacen los virus. ¿Qué objetivos pueden perseguir aquellas personas que programan viruses? Como dato de color y para relacionar con lo visto en la clase 9, se les puede contar que, en general, muchos de los viruses son escritos en lenguaje de máquina. Además, para descubrir vulnerabilidades, los *hackers* suelen desensamblar el archivo binario de un programa (usando herramientas como BARF) para ver cómo está hecho y analizar dónde puede haber un agujero de seguridad.

Para la próxima actividad se dividirán en grupos de 3 personas y a cada grupo se les dará un artículo muy breve acerca de algún caso de un software malicioso que haya tenido cierto impacto. Se sugiere proponer casos de los siguientes tipos de programas maliciosos:

- Troyano
- Keylogger
- Ransomware[33]
- Spyware
- Stealer
- Puerta trasera o *backdoor*
- Bomba lógica

Luego de que cada grupo lea su artículo, se realizará una puesta en común en donde cada equipo contará al resto las características del software malicioso que les tocó. En cada caso, deberían poder identificar qué hace y cómo se transmite.

La siguiente actividad abordará un tema de suma importancia a la hora de utilizar Internet: la suplantación de identidad o *phishing*. Como primera aproximación se le mostrará a la clase la siguiente captura de pantalla y se les preguntará si notan algo extraño en ella:

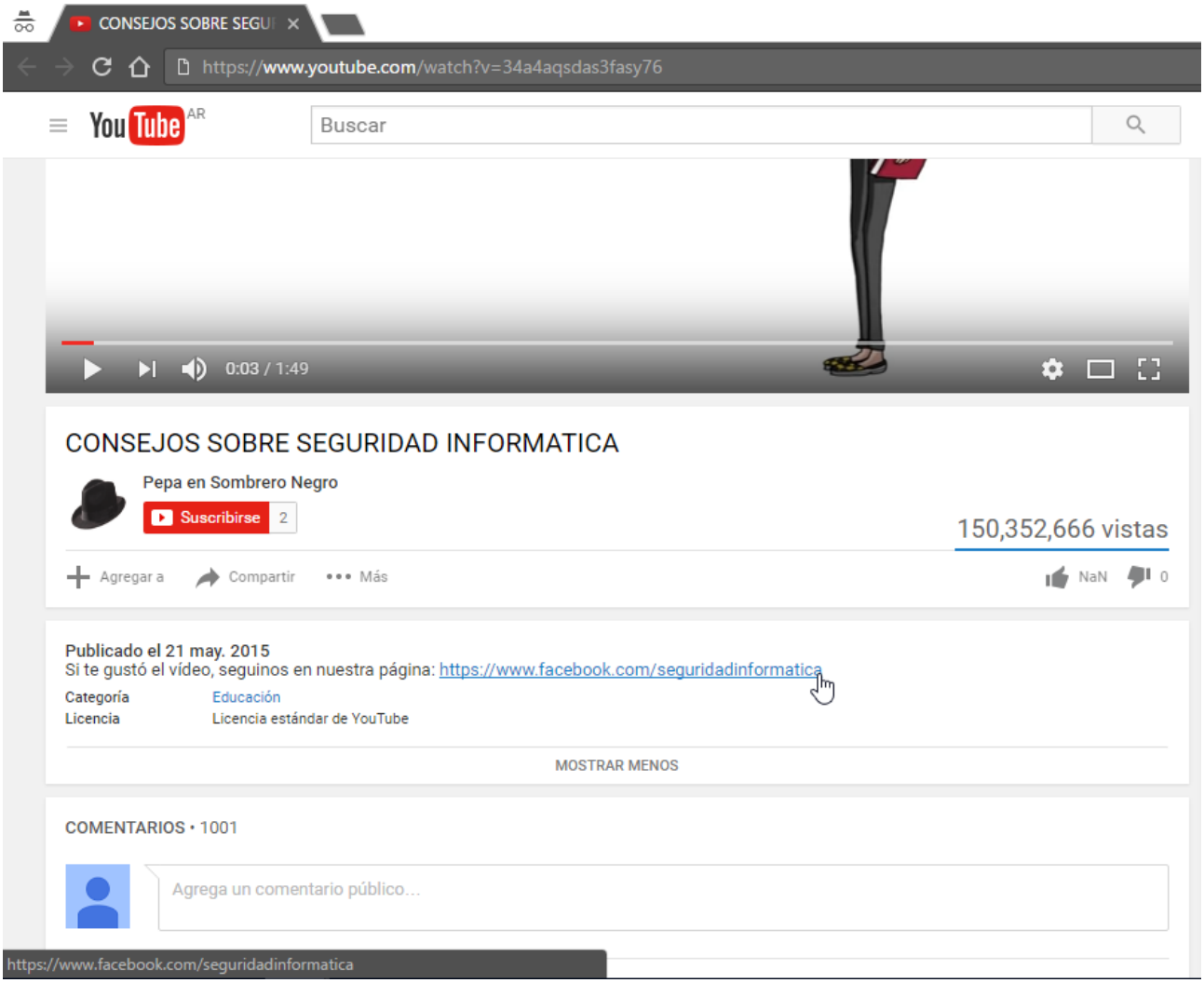
[33] Se puede trabajar la historia del argentino José Vildoza, quien descubrió una vulnerabilidad en un ransomware que afectó la computadora de su padre logrando anular el efecto malicioso de dicho programa.

El problema que tiene es que el sitio al que queremos entrar, Facebook, no es al que estamos entrando, que es Fakebook. Esto se puede apreciar observando la URL o dirección del sitio: https://www.fakebook.com.

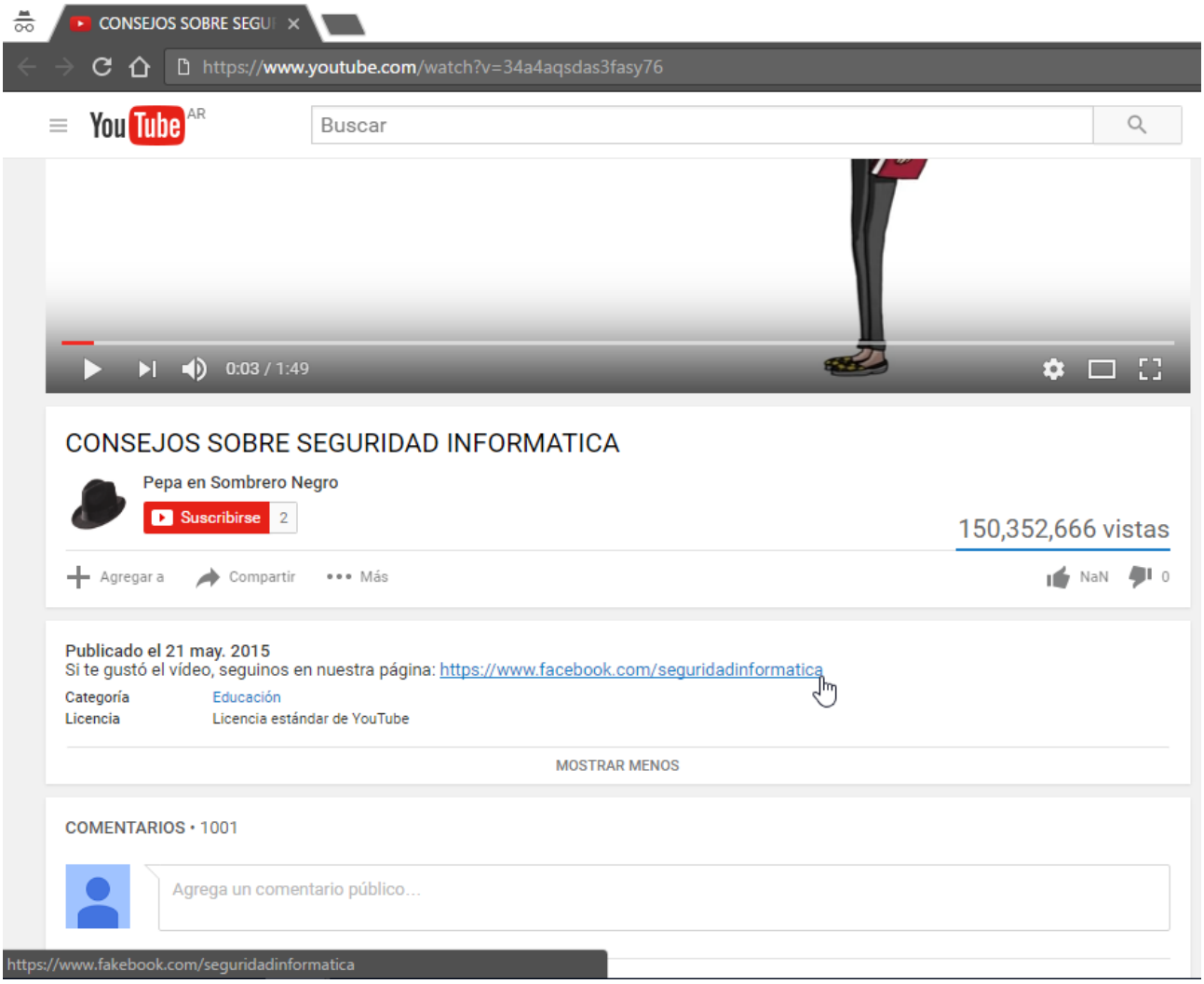
- ¿Por qué alguien haría un sitio idéntico a Facebook pero con otra dirección?
- ¿Cómo pudimos haber llegado a Fakebook?
- ¿Qué podría pasar al intentar ingresar a Fakebook?
- ¿Cómo podríamos evitar ser engañados en estos casos?

Para ahondar un poco más en esta problemática se les presentarán los siguientes 3 escenarios, dos de los cuales buscan realizar una suplantación de identidad y el otro es legítimo. Deben decidir cuál es el legítimo y cuáles no y, en cada caso, por qué.

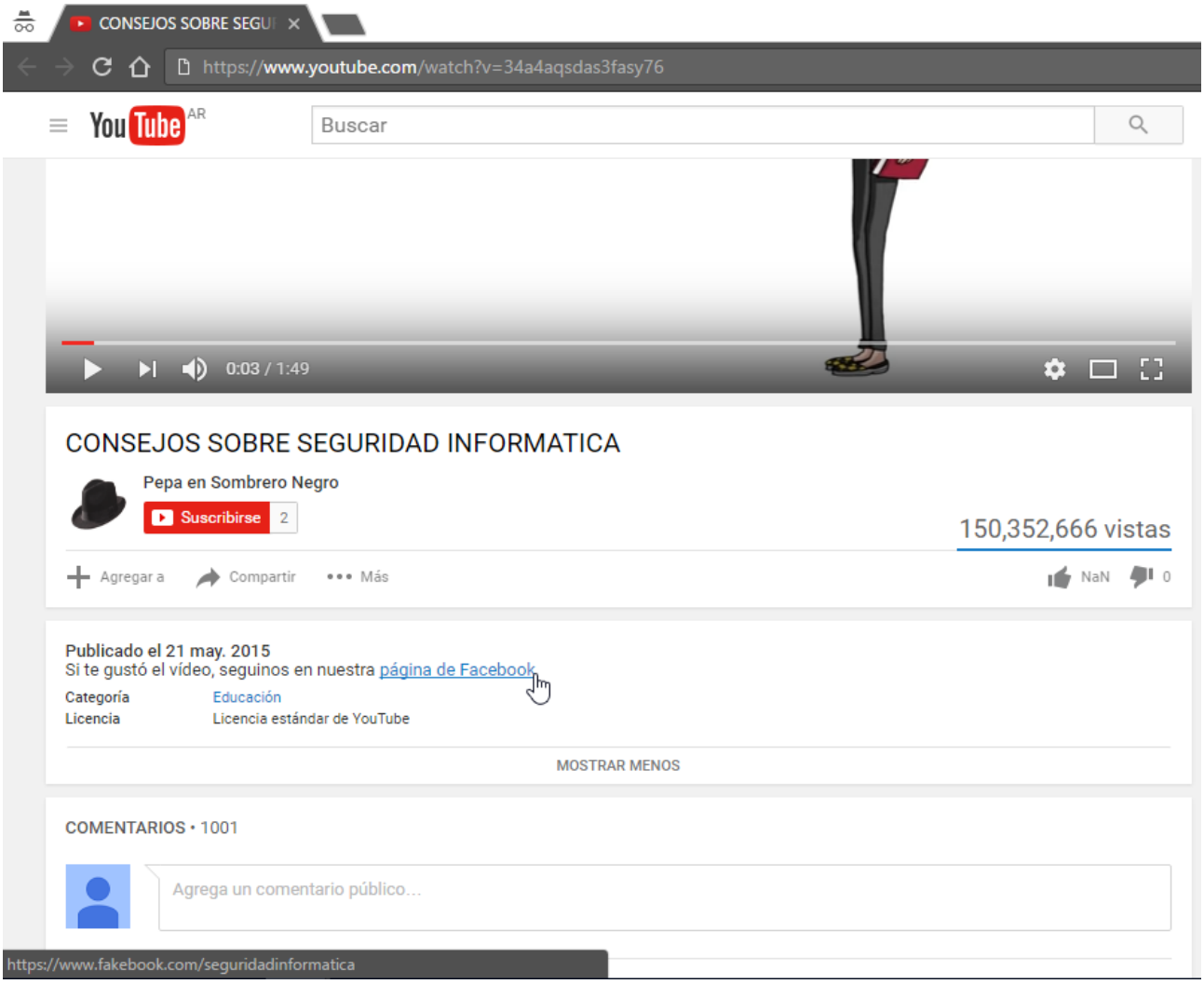
Escenario 1:



Escenario 2:


CONSEJOS SOBRE SEGU
https://www.youtube.com/watch?v=34a4aqsdas3fasy76
YouTube AR
Buscar
0:03 / 1:49
CONSEJOS SOBRE SEGURIDAD INFORMATICA
Pepa en Sombrero Negro
Suscribirse 2
150,352,666 vistas
Agregar a
Compartir
Más
NaN
0
Publicado el 21 may. 2015
Si te gustó el vídeo, seguinos en nuestra página: https://www.facebook.com/seguridadinformatica
Categoría Educación
Licencia Licencia estándar de YouTube
MOSTRAR MENOS
COMENTARIOS • 1001
Agrega un comentario público...
https://www.fakebook.com/seguridadinformatica

Escenario 3:



Se espera que las y los estudiantes identifiquen la diferencia entre un texto linkeado y el link al que dirige ese texto. En los navegadores web, esto se puede verificar posándonos con el mouse encima del link y, sin cliquear en él, abajo a la izquierda se muestra en un cartelito negro cuál es el link que efectivamente se va a abrir. Es muy importante que noten que el texto que cliqueamos puede no tener nada que ver con el link a donde se nos va a dirigir. Esta modalidad es sumamente común a la hora de realizar engaños o fraudes en Internet ya que las personas generalmente no chequean el link que van a abrir.

Teniendo esto en cuenta, se puede concluir que el escenario 1 es legítimo y los escenarios 2 y 3 pretenden realizar una suplantación de identidad.

Por último se sugiere dedicar el final de la clase a abordar distintas maneras en que los usuarios podemos protegernos frente a los programas maliciosos:

- Instalar las actualizaciones del sistema operativo y de los programas. Estas corrigen defectos que son los que usan los virus para entrar a las computadoras.
- Evitar la autoejecución de los programas y la autodescarga.
- Utilizar un antivirus tanto en la computadora como en el celular.
- Utilizar un Firewall.
- Analizar los permisos que requieren las aplicaciones que instalamos en el celular. Por ejemplo, ¿sería razonable que la aplicación de una linterna requiriera acceder a todos nuestros contactos? ¿Para qué piensan que utilizan esa información?
- Utilizar normalmente un usuario con permisos menores a Administrador y, solo en casos particulares, por ejemplo al querer instalar un programa, habilitar dicho modo.
- Hacer copias de seguridad (también llamadas *backups*) con periodicidad.
- Tener cuidado con los sitios que visitamos y los links que cliqueamos.
- Utilizar contraseñas seguras. Suele ser más importante que la contraseña sea larga a que sea complicada de acordarse. La dificultad para adivinar una contraseña crece cuanto más grande sea ésta. Una buena idea es poner como contraseña una frase sin sentido pero que podamos recordar. Por ejemplo: *20elefantesgalacticosdelmedioevo*.
- Siempre tener en mente el grado de confianza que tenemos en el sitio que visitamos, el programa que estamos usando, la persona con la que estamos hablando por chat, etc.

Como tarea para realizar en casa, se puede pedir a la clase que investiguen y realicen un breve informe sobre las siguientes herramientas de seguridad informática, explicando brevemente qué son y cómo funcionan:

- Firewalls
- Antivirus
- Adblockers
- Certificados HTTPS
- Cifradores de mail

Clase 21 - Repaso de programación

Se retomarán los contenidos vistos con anterioridad (con énfasis en las clases 1 a 7) a través del siguiente ejercicio: se les pedirá a las y los estudiantes que programen el juego "Adivinador" de dos jugadores, en el cual uno deberá adivinar el número ingresado por el otro.

El juego comenzará pidiéndole a cada jugador su nombre. A continuación el primer jugador deberá ingresar un número (del 1 al 5) -sin que el otro mire el teclado- y el segundo jugador intentará adivinarlo, ingresando a su vez otro número. Luego, la computadora se encargará de verificar si ambos números coinciden. Si se produce una coincidencia ganadora la computadora dirá el nombre del segundo jugador y mostrará en pantalla el número adivinado. En cambio, si no se produce una coincidencia, se proclamará como ganador al primer jugador y también se mostrará en pantalla el número que no ha sido adivinado[34].

Este ejercicio pone el énfasis en el trabajo con variables, debido a que sólo es posible comparar los números ingresados si la computadora los tiene guardados en memoria para poder acceder a ellos. Por otro lado introduce una idea previamente trabajada: las operaciones sobre expresiones, a partir de la evaluación de si un número es mayor que otro.

Se espera que luego de haber trabajado el tiempo necesario en el ejercicio, se ponga énfasis en que el resultado de la condición evaluada varía en cada ejecución. Al momento de crear el programa no es posible conocer si los números ingresados por los jugadores coinciden o no. Es por ello que resulta indispensable programar dos caminos alternativos que modificarán el desenlace del juego a partir de la estructura del “Si… Entonces…” (sea que el segundo jugador adivine o no el número).

Al final de esta actividad se les indica a las y los estudiantes que guarden el proyecto ya que luegos será modificado.

Como cierre de la actividad, se introduce el manejo de números aleatorios en Alice, a partir de un breve ejemplo en el que un número aleatorio dado por la computadora se imprime en un texto 3D. Al ejecutar reiteradas veces el juego el número se irá modificando.

Se introducirán cambios en el juego anterior. Se les propone a las y los estudiantes modificar el programa anterior para que se batan a duelo la computadora y un único jugador. La computadora definirá un número al azar (del 1 al 5, sin decimales) y el jugador deberá adivinarlo. Si lo adivina la computadora se dará por derrotada mostrando el número en pantalla. Si no lo adivina la computadora mostrará un mensaje victorioso y el número que no fue adivinado.

Luego de que las y los estudiantes hayan podido dedicar un tiempo a avanzar con el ejercicio, se espera reservar 20 minutos finales de la clase a la realización en conjunto de este último ejercicio.

---

[34] Se deberá explicar de manera adicional la función “[what] como secuencia de caracteres” y el uso de texto 3D.

Clase 22 - Métodos II

La siguiente actividad permitirá reforzar el uso adecuado de métodos y alternativas condicionales, sumado a la interacción con las y los usuarios. Se continuará investigando la creación de métodos propios a partir de la siguiente consigna.

Se les presentará a las y los estudiantes una historia similar a la trabajada en “Encuentro de tercer tipo”, pero esta vez la reacción del robot al descubrir vida en otro planeta dependerá del usuario, al estilo *Elige tu propia aventura*. La historia se encuentra disponible en el documento: “Guión del encuentro de tercer tipo. Elige tu propia aventura”. (Ambos archivos deberán ser compartidos a las y los estudiantes a través de un pendrive o de Internet). Para comenzar la actividad se leerá en conjunto y voz alta la historia:

Después de viajar por el espacio, una nave tripulada por un robot acaba de aterrizar en la Luna. El robot instala en la superficie una cámara que nos permite ver la escena: el robot, la nave espacial y la Luna. De repente un extraterreste aparece de atrás de algunas rocas, y dice "Bienvenidx". Aquí la escena se congela, y aparece una pregunta al usuario: ¿es el alien amigable (opción 1) o temible (opción 2)? Si el usuario indica el número 1, es decir que el alien es amigable: el robot gira su cabeza, se acerca al extraterrestre y lo saluda. Si el usuario indica el número 2, es decir que el alien es temible: el robot gira su cabeza, mira a cámara y huye rápidamente a la nave.

Se les propondrá a las y los estudiantes que comiencen a programar el guión.

Se espera que a partir de esta clase, las estrategias de las y los estudiantes para la resolución de las consignas tenga en cuenta la división del problema en subtareas, la legibilidad de los programas y la definición de nombres de métodos pertinentes. Es por ello que se insiste con la puesta en común de los programas y en la explicación colectiva de las soluciones propuestas por las y los estudiantes.

Como cierre se compararán algunas soluciones posibles buscando discutir colectivamente sobre qué tan apropiadas son las abstracciones usadas y sus nombres.

Clase 23 - Proyecto integrador I

Presentación de la consigna y trabajo grupal inicial. Se trata de un proyecto en el que puedan poner en juego los conceptos de programación aprendidos hasta ahora.

Comenzar el trabajo en clase.

Clase 24 - Métodos III

Se divide a las y los estudiantes en cuatro grupos y se les lee un enunciado (como el que sigue). Se les entrega a cada grupo 6 programas como soluciones factibles. Cada grupo deberá debatir y argumentar internamente cuál es el programa que considera resuelve de manera más adecuada el problema. Una vez hecho esto, cada grupo pasa al frente y explica qué solución eligieron y por qué.

Luego se pone en conjunto la solución que el/la docente considera correcta argumentando las razones.

Ejemplo de enunciado:

Se desea implementar un juego sencillo en el que dos personajes compiten una carrera, con dos particularidades: la primera es que los personajes se mueven dando saltos. La segunda, es que en cada momento se ejecuta sólo una acción, de manera tal de que cada jugador debe elegir entre hacer avanzar a su personaje o hacer retroceder al otro.

La tecla → hace avanzar al canguro, mientras que la tecla ← hace retroceder a la ardilla, y la tecla A hace avanzar a la ardilla, mientras que la tecla D hace retroceder al canguro.

Los archivos a analizar para este ejemplo se encuentran en https://goo.gl/5d4HWe.

En la primera solución se puede ver que no hay ningún tipo de abstracción, resultando en un código de muy dificultosa lectura. La segunda presenta cierto nivel de abstracción, aunque resulta en un nivel de agrupamiento insuficiente (hay métodos para las partes que componen un salto, pero no para el salto en sí mismo).

La tercera opción se parece a la segunda, salvo porque los nombres de los métodos son peores, mucho menos declarativos. La cuarta es aún peor, ya que métodos que realizan acciones similares en la ardilla y en el canguro reciben nombres distintos.

La quinta está basada en la segunda e incorpora el método saltar para ambos personajes, lo que permite entender a alto nivel el código sin entrar en detalles. Sin embargo, la selección de ganadores aún no está abstraída de ninguna manera. La sexta solución corrige este problema.

Clase 25 - Repetición simple

En esta clase se continuará desarrollando el juego de adivinación de números aleatorio visto en la clase 21. Para ello, antes de introducir el tema nuevo, se les pide abrir nuevamente dicho programa. Dado que es difícil adivinar el número, se les indica a las y los estudiantes que complejicen el juego dándole 3 oportunidades al jugador para adivinarlo.

Se espera que las y los estudiantes comiencen a duplicar las instrucciones una a una, para abarcar los múltiples intentos. En algunos casos es posible que se utilice el recurso de los métodos (por ejemplo para evaluar la coincidencia de los números).

Una vez transcurrido un tiempo para que las y los estudiantes prueben diferentes estrategias de solución del problema, se les pide que el rango de números a adivinar sea del 1 al 10 y que el jugador tenga 5 oportunidades.

Se esperará el tiempo suficiente para que las y los estudiantes identifiquen la necesidad de un recurso faltante frente a las tareas repetitivas que implican hacer el programa.

Una vez transcurrido un tiempo para que las y los estudiantes prueben diferentes estrategias de solución del problema, se les proponen las siguientes preguntas: si les dijera que ahora la cantidad de oportunidades asciende a 20, ¿cómo resolverían el problema? ¿Con esa solución cuán extenso sería el programa?

Se observa en conjunto la dificultad de ampliar el programa a un número elevado de intentos. A continuación, se les presenta una nueva estructura de repetición, el *loop* o *ciclo*, que permite repetir una cantidad de veces determinada una o más instrucciones, que se escriben una sola vez. Se les indica dónde encontrar el *loop* y se los dejará investigar su utilización.

Se les pedirá que adapten el juego para incluir 5 intentos, utilizando un *loop* con ese número de repeticiones.

Como cierre de la actividad, se deberá volver sobre la diferencia del *recorrido* entre un programa secuencial y uno que utiliza ciclos. Por otro lado, retomar la solución que provee el *loop* para lograr la repetición de una o más instrucciones un número de veces conocido y compartir entre todos su utilidad frente a situaciones en las que es posible identificar que un número de pasos deben ser ejecutados una cantidad específica de veces.

Por ejemplo, ¿qué acciones cotidianas nos obligan a repetir nuestros movimientos una cantidad conocida de veces? Se les pide a las y los estudiantes que simulen hacer doble clic: ¿es una cantidad de repeticiones fija? ¿Y subir una escalera de 25 escalones?

Clase 26 - Variables II

El objetivo es la actividad es implementar un contador de puntaje para un jugador de un juego simple, que comience en cero y se incremente o decremente en función del accionar del jugador. Esta actividad le permite a las y los estudiantes comenzar a introducirse en un uso de las variables como contadores.

Se presenta el escenario: Juego de feria. En él habrá una liebre que presenta el juego, un mostrador con 3 pilas de conos y un guante lanzador de bolas. En cada jugada la computadora decide al azar cuál de las 3 pilas estará disponible para ser derribada. Previamente se le pide al jugador que elija una de las pilas, sin conocer cuál de ellas será la única disponible. De acuerdo al número que indique el jugador la mano lanzará una bola a esa pila de conos.

A partir de este funcionamiento inicial se espera que las y los estudiantes finalicen la dinámica del juego programando el puntaje del jugador.

Inicialmente si coinciden ambos números (la pila de conos disponibles y la pila hacia dónde se lanzó la bola), el jugador logrará el objetivo de derribar una pila de conos y se obtendrá un puntaje de 1. La liebre entonces indicará que el puntaje es 1.

Una vez lograda esta consigna se les pedirá a las y los estudiantes que complejicen el funcionamiento del puntaje de la siguiente manera. Al comenzar el programa nuestro puntaje siempre debe ser 0. En cada ejecución del programa tendremos 3 oportunidades para acertar. Si acertamos un lanzamiento sumaremos un punto a nuestro puntaje. La liebre irá indicando el puntaje nuevo luego de cada lanzamiento.

A partir de esta segunda consigna se dejará a las y los estudiantes retomar el programa anterior. La propuesta incluye retomar el trabajo con alternativas condicionales y variables, ya que se espera que el puntaje se contabilice dentro de una variable que comenzará siempre en 0, se incrementará en uno cada vez que se produzca un acierto y si no se produce un acierto mantendrá su valor anterior.

Por último, se les pedirá a las y los estudiantes que guarden el programa ya que lo usarán nuevamente en la clase 27.

Como cierre de la clase se terminarán de pulir los últimos detalles del proyecto integrador del juego presentado en la clase 23.

Clase 27 - Repetición condicional

La clase se inicia retomando el trabajo con variables del programa Juego de feria (clase 26).
Se les pide a las y los estudiantes que continúen con la programación del juego para que finalice cuando el jugador acierta 3 lanzamientos. El jugador comenzará con un puntaje de 0 y podrá jugar las veces que sean necesarias hasta lograr 3 aciertos. La liebre irá indicando el puntaje del jugador y cuando llegue a 3 puntos lo declarará ganador y el juego terminará.

Se espera que las y los estudiantes exploren una solución a partir de una repetición simple u otras estrategias conocidas.

Luego de un lapso razonable de tiempo de experimentación se les pregunta a las y los estudiantes:

- ¿Cuántas veces jugamos hasta que se acaba el juego?
- ¿Qué información nos permite saber si perdimos y no debemos continuar jugando?

La respuesta deberá ser algo similar a: debemos jugar mientras nuestro puntaje sea menor a 3.

Dando pié a lo ya conocido se les pedirá que recuerden el uso de la repetición definida:

- ¿Es posible decir cuántos estudiantes hay hoy en el aula?
- ¿Y por qué no podemos indicar cuántas veces debo jugar hasta ganar?
- ¿Cúal es la diferencia entre ambas cantidades?

La dificultad de este ejercicio es que no se conoce la cantidad exacta de veces que el jugador jugará una partida hasta ganar. Pueden ser 10 veces, 20, 30, etc.

Aquí entra en escena el "while" de Alice, que significa "repetir mientras que...". Ese bloque repite todas las instrucciones que tenga adentro, mientras se siga cumpliendo una condición, en este caso mientras que el puntaje del jugador sea menor a 3.

Finalmente se propone detenerse en la diferencia del recorrido propuesto por la alternativa condicional, en el cual la condición era evaluada una única vez para tomar uno de los dos caminos alternativos. En cambio, la repetición condicional vuelve una y otra vez sobre la condición (¿mi puntaje es igual a tres?) para decidir el camino a seguir (sea seguir jugando o finalizar el juego).

Como cierre, se les propone a las y los estudiantes modificar el programa para que el jugador únicamente tenga 5 oportunidades de lanzamiento, si en ellas logra 3 aciertos será declarado por

la liebre como ganador, lo que implicará introducir otra variable sobre la que establecer la condición.

Clase 28 - Proyecto integrador II

En esta clase se continuará con el trabajo en el proyecto final, aprovechándose para resolver dudas y consultas, y monitorear el progreso de cada grupo.

Se fomentará que las y los estudiantes exploren los proyectos de sus compañeros y debatan entre sí las dudas y consultas antes de recurrir al docente.

Se mencionará además que en la clase siguiente, además de entregar el proyecto, deberán presentarlo brevemente de manera oral. La presentación consistirá en una sección de máximo de 5 minutos donde se lo muestra, y otra de máximo de 10 minutos donde se explicará cómo fue realizado, no de manera exhaustivo sino haciendo énfasis en las partes más interesantes o dificultosas. El/la docente podrá sugerir a cada grupo qué parte exponer.

Clase 29 - Proyecto integrador III

En esta clase los distintos grupos presentarán sus proyectos, debiendo mostrar el funcionamiento pero también explicar cómo fueron realizadas algunas partes que resulten particularmente interesantes o dificultosas. Luego de las exposiciones se habilitará una instancia de preguntas y respuestas entre las y los estudiantes.

Bibliografía

Material didáctico

- Dale Aceptar
- Dann W., Cooper S. y Pausch R., “Learning to Program with Alice”, Prentice Hall, 2012.
- Guía de actividades prácticas en Alice, UNC++, 2015.
- Repositorio de materiales y actividades en Alice (en inglés).
- Planificaciones de clases usando Alice (en inglés).
- Factorovich, P. M. y Sawady, F. A., “Actividades para aprender a Program.AR”, Fundación Sadosky, 2015.
- Proyectos de Enseñanza de Programación para la Escuela Primaria, UNC++.
- Unidades didácticas sobre seguridad informática.
- Bell, T. C., Witten, I. H., Fellows, M. R., Adams, R. y McKenzie, J., “CS Unplugged: An Enrichment and extension programme for primary-aged students”, 2015.
    - Versión en español.
    - Versión en inglés (incluye más actividades, es la versión más actualizada).
    - Videos de actividades (en inglés).
- “Computer Science Field Guide (Teacher Version)”, CS Education Research Group, Universidad de Canterbury, Nueva Zelanda, 2017.
- “Computer Science Field Guide (Student Version)”, CS Education Research Group, Universidad de Canterbury, Nueva Zelanda, 2017.
- Mumuki.org
- Code.org

Material ampliatorio

- Tanenbaum, A. S., “Organización de computadoras. Un enfoque estructurado”, Prentice-Hall, 2000.
- Stallings, W., “Organización y arquitectura de computadores”, Prentice-Hall, 2006.
- Null L. y Lobur J., “Essentials of computer organization and architecture”, Jones & Bartlett Learnin, 2015.
- Tanenbaum, A. S., “Sistemas operativos modernos”, Prentice-Hall, 2009.
- Silberschatz A., Gagne G. y Galvin P. B., “Fundamentos de sistemas operativos”, McGraw-Hill, 2006.
- P. Martínez López, “Las bases conceptuales de la Programación. Una nueva forma de aprender a programar”, La Plata, Argentina, 2013.

Software

Pilas Bloques
Alice
Scratch
Gobstones
App Inventor

Créditos

Autores

(por orden alfabético)
Teresa Alberto
Fernando Schapachnik
Herman Schinca
Daniela Villani

Diseño gráfico e ilustración

Jaqueline Schaab

Coordinación Iniciativa Program.AR

María Belén Bonello
Pablo Factorovich
Fernando Schapachnik
www.program.ar

Autoridades Fundación Dr. Manuel Sadosky

Presidente: Dr. Lino Barañao
Director Ejecutivo: Dr. Esteban Feuerstein
www.fundacionsadosky.org.ar

Esta obra está bajo Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.