UTPL
La Universidad Católica de Loja

Modalidad Abierta y a Distancia

Fundamentos de programación

Guía didáctica

Índice
Primer bimestre
Segundo bimestre
Solucionario
Referencias bibliográficas

Índice
Primer bimestre
Segundo bimestre
Solucionario
Referencias bibliográficas

UTPL

Departamento de Ciencias de la Computación y Electrónica

Sección Tecnologías Avanzadas de la Web y SBC

Fundamentos de programación

Guía didáctica

Autor:

Jorge Afranio López Vargas

DSOF_1073

Fundamentos de programación
Guía didáctica
Jorge Afranio López Vargas

Universidad Técnica Particular de Loja

Diagramación y diseño digital:
Ediloja Cía. Ltda.
Telefax: 593-7-2611418.
San Cayetano Alto s/n.
www.ediloja.com.ec
edilojainfo@ediloja.com.ec
Loja-Ecuador

ISBN digital - 978-9942-25-721-5

24 de abril, 2020

Índice

1. Datos de información

1.1. Presentación. Orientaciones de la asignatura

1.2. Competencias genéricas de la UTPL

Organización y planificación del tiempo

1.3. Competencias específicas de la carrera

- Construir modelos específicos de ciencias de la computación mediante esquemas matemáticos y estadísticos, para propiciar el uso y explotación eficiente de datos e información.

- Implementar aplicaciones de baja, mediana y alta complejidad integrando diferentes herramientas y plataformas para dar solución a requerimientos de la organización.

1.4. Problemática que aborda la asignatura en el marco del proyecto

Dentro de la carrera de Tecnologías de la Información, el desarrollar las competencias necesarias que permitan a sus estudiantes resolver problemas y expresarlos como un programa computacional con un conjunto de características que lo transforman en una solución optima, además de identificar las tecnologías de información y el usó, son actividades fundamentales que aportan directamente al desarrollo del perfil profesional y su campo ocupacional.

Apoyar a desarrollar esas competencias es justamente el objetivo principal de la asignatura de Fundamentos de Programación. Con las competencias que adquirirá al finalizar la asignatura, estará en la posibilidad no solo de aprender a programar en seudocódigo, sino de usar un lenguaje de programación de alto nivel y sus herramientas para escribir, compilar y ejecutar esos programas en un computador, además empleará técnicas, buenas prácticas y estrategias para que sus programas sean eficientes, legibles y correctamente documentados. Así como también tendrá la capacidad de reconocer, clasificar y determinar cómo se utilizan las tecnologías de la información en su entorno cotidiano.

2. Metodología de aprendizaje

La metodología que se utiliza se denomina blended learning.

A través de esta metodología el proceso de enseñanza aprendizaje se divide en trabajo autónomo y actividades síncronas. Lo que le permite al estudiante organizar su tiempo y actividades para cumplir con las actividades propuestas en cada una de las asignaturas. Además, permite que el docente acompañe al estudiante a través de diferentes espacios, tales como de chat de tutoría permanente, foro académico y chat académico que le brindan al estudiante la posibilidad de aclarar cada una de sus dudas, además de recibir un trato personalizado que le ayuda a avanzar en la adquisición de las competencias de esta asignatura.

Más información la puede encontrar aquí: https://aretio.hypotheses.org/2437

3. Orientaciones didácticas por resultados de aprendizaje

Primer bimestre

Resultado de aprendizaje

Utiliza tipos de datos primitivos y estructuras de datos

Contenidos, recursos y actividades de aprendizaje

Antes de iniciar con el desarrollo de su proceso de aprendizaje, es oportuno aclarar que *Fundamentos de Programación* es la continuación de la asignatura denominada *Algoritmos y Resolución de Problemas*. Tomando en cuenta esta característica, lo invito a utilizar todos los conceptos y herramientas que aprendió en dicha asignatura y utilizarlos aquí ya que se llevará esos conceptos, ya asimilados, un paso hacia adelante, la creación de programas utilizando un lenguaje de programación real.

En esta primera semana aprenderá sobre los conceptos fundamentales de programación, aquellos que están presentes en la mayoría de los lenguajes de programación actuales. Al concluir la

semana usted podrá escribir programas relativamente simples que permitan resolver algunas fórmulas o ecuaciones matemáticas.

Para conseguir lo anterior es necesario que revise los fundamentos teóricos sobre cada uno de los temas, para luego asociarlos con un lenguaje de programación y finalmente poner en práctica lo estudiado, resolviendo un problema a través de la construcción de un programa.

Dentro de los fundamentos teóricos existe la guía didáctica López Vargas, J. (2018) *Guía Didáctica de Fundamentos de Programación*. Loja – Ecuador: Editorial UTPL, que le señalará que recursos revisar y cuando revisarlos, además de hacer varias puntualizaciones que son importantes para el desarrollo de cada uno de los contenidos. También utilizará el libro base López Román, L. (2013) *Metodología de la Programación Orientada a Objetos*. México. (2da edición). México: Alfaomega Grupo Editor.

El aprendizaje del lenguaje de programación se hace utilizando el recurso Lopez-Vargas, J. (2019) *Java, Apuntes básicos*. Recuperado de: http://j4loxa.com/courses/java101/Java_Apuntes_Basicos. epub, el cual le ayudará a entender como cada uno de los temas que estudió teóricamente se plasman en un lenguaje de programación. Así mismo es necesario que siga el documento, revise los recursos que propone y desarrolle las actividades que señala, según las instrucciones que encontrará en este entorno virtual de aprendizaje.

El entorno virtual de aprendizaje se convierte en un director que señala y delega el desarrollo de los contenidos a los documentos comentados anteriormente. Además, aquí encontrará un resumen de cada tema, siempre que sea posible relacionando lo aprendido en Algoritmos y Resolución de Problemas, mini-especificaciones, el seudocódigo que se propone y el lenguaje de programación seleccionado.

El lenguaje de programación seleccionado es el lenguaje de programación Java, que es un lenguaje de alto nivel y que desde hace algunos años es el más utilizado en el desarrollo de diferentes tipos de aplicaciones, tales como móviles, Web y de escritorio.

Semana 1

Unidad 1. Conceptos fundamentales de programación.

1.1. Conceptos básicos de programación

Considerando que la programación es el acto de diseñar e implementar programas computacionales, Horstmann, (2016). En esa definición existen varios conceptos que son importantes estudiar. Dos de ellos son computadora y programa, es por ello que es necesario que realice la siguiente lectura.

"1.1 Conceptos generales" del texto base (López, (2013)), realizar su lectura hasta el tema "Estructuras de Control" (incluido).

¿Encontró los elementos que forman un programa? Estoy seguro de que si, también confío que pudo asociarlos con los conceptos que estudió en la asignatura Algoritmos y Resolución de Problemas.

Resulta sorprendente un programa, que puede verse como algo complejo en realidad utilice tan pocos elementos para su funcionamiento. Más adelante centrará su atención en cada uno de estos elementos.

Ahora es necesario entender como un computador ejecuta un programa. Con este fin es menester que revise el siguiente recurso: *Loya, J. (2014) ¿Cómo se ejecuta un programa? Recuperado de* https://goo.gl/wRrrcj.

Recuerde que los programas antes de ser ejecutados deben ser cargados en memoria, es decir cada una de las instrucciones que forman un programa se encuentran en la memoria del computador y son recuperadas, decodificadas y ejecutadas.

Hasta el momento conoce qué es un programa, un computador y como los primeros se ejecutan en los segundos. De seguro se está preguntando ¿cuál es el proceso por seguir para elaborar programas? Para dar respuesta a esa pregunta realice la siguiente lectura:"1.3 El proceso de programación" del texto base (López, (2013)).

¿Pudo determinar en qué paso se ubica lo aprendido en Algoritmos y resolución de problemas? Estoy seguro que sí; sino fue así es necesario que ponga especial énfasis en el tercer paso. De seguro se encontró con conceptos ya conocidos como algoritmos.

Antes de cerrar este primer apartado es necesario analizar con un poco más de detalle el proceso de Codificación. En este paso se utilizan los lenguajes de programación para crear un programa que ha sido previamente diseñado, pero ¿Qué es un lenguaje de programación? ¿Qué características tiene? ¿Qué modelos siguen los lenguajes de programación? Para dar respuesta a estas preguntas, es necesario realizar la siguiente lectura: "1.1 Conceptos generales" del texto base (López, (2013)), realizar su lectura desde el tema

"Lenguajes de programación" hasta el tema "1.3 El proceso de programación".

Ahora que ya conoce las generalidades de los lenguajes de programación, es conveniente empezar a conocer el lenguaje de programación que aprenderá en el transcurso de esta asignatura, para cumplir con este objetivo es necesario que revise el siguiente recurso educativo: "Java y sus características" que está disponible en Lopez-Vargas, (2019).

Es necesario avanzar y revisar cada uno de los elementos que forman un programa, no olvide que estos conceptos son sumamente importantes ya que son la base para el contenido que se desarrolla posteriormente, así que le recomiendo revisar con mucho cuidado los contenidos que se presentan a continuación.

1.2. Estructura de los datos

Previamente conoció que un lenguaje de programación está compuesto por 3 elementos, uno de esos elementos son las estructuras de datos. Este tema, aunque le parezca nuevo, no lo es ya que en la asignatura Algoritmos y Resolución de problemas lo estudió bajo el nombre de Variables y tipos predefinidos.

La base teórica de este tema se encuentra en el documento López Vargas, J. (2018) Guía Didáctica de Fundamentos de Programación. Loja – Ecuador: Editorial UTPL. Revise el tema: "1.7 Estructura de datos."

¿Ha podido relacionar estos conceptos con los que ya estudió previamente? Seguro que fue así. Si tiene alguna duda le propongo que revise la Figura 2 que resume este tema abordado desde tres diferentes enfoques. El primero, la mini-especificación, muestra tanto el nombre del tipo de dato, como su símbolo abstracto; el segundo, el seudocódigo, el que se utiliza en el texto base y

finalmente, el lenguaje de programación Java, que aprenderá en esta asignatura y que empezó a estudiar anteriormente. Los guiones señalan la ausencia de un tipo de dato en ese enfoque.

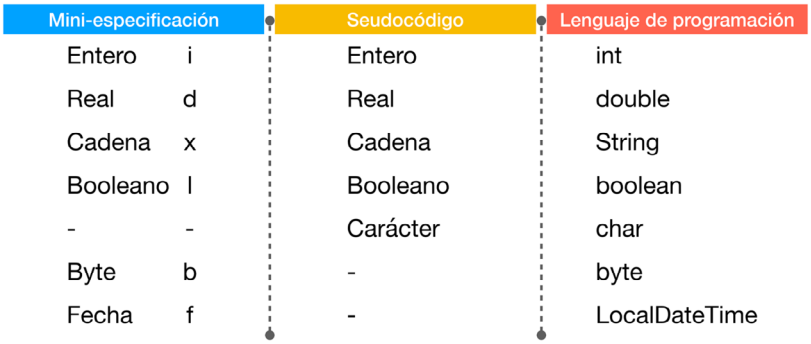


Figura 1. Tipos de datos desde 3 enfoques diferentes

Los nombres de variables y constantes, además de seguir un estándar, deben ser seleccionados de tal manera que representen la función que desempeña dentro del programa, pero en especial que revelen por qué existen, qué es lo que hacen y cómo usarlas, Martin, (2009). Para aclarar esto, analice el siguiente ejemplo: en un programa se necesita una variable que represente al primer apellido de una persona ¿qué nombre seleccionaría? La Tabla 3 muestra un listado de opciones y su evaluación utilizando dos criterios, a) el nombre es representativo y b) cumple el estándar de programación del lenguaje Java.

Tabla 1. Evaluación del nombre de una variable

Posibles nombres	Representativo	Estándar de programación
pa	No	No. No es una palabra
1er_apellido	Si	No. Deben empezar por una letra
PRIMERAPELLIDO	Si	No. Se debe escribir en minúsculas.
primerapellido	Si	No. En caso de estar formado por más de una palabra, la primera letra de la segunda palabra va en mayúscula.

Posibles nombres	Representativo	Estándar de programación
Primerapellido	Si	No. Debe iniciar con una letra minúscula.
PrimerApellido	Si	No. A partir de la segunda palabra la primera letra se escribe en mayúscula.
apellido1	Si	Si. Pero existe una mejor alternativa.
$primerApellido	Si	No. Debe iniciar con una letra.
primerApellido	Si	No. Debe iniciar con una letra
primerApellido	Si	Si. Inicia con una palabra en minúscula y la primera letra de la segunda palabra se escribe en mayúscula.

Es momento de avanzar en el desarrollo de los contenidos, hasta el momento ha estudiado tan solo uno de los elementos que forman un programa, las variables. La siguiente sección muestra algunas operaciones primitivas que se pueden hacer con las variables.

1.3. Operaciones primitivas elementales

Para poder escribir programas, los lenguajes de programación en general poseen un grupo de instrucciones que formalmente se denominan sentencias. Las primeras sentencias que aprenderá se describen en López, (2018) en el apartado “1.3 Operaciones primitivas elementales”

Una vez realizada la lectura trate de responder a la pregunta ¿Qué tipos de programas se puede construir con estas instrucciones primitivas elementales?

De seguro está ansioso por relacionar lo aprendido con el lenguaje de programación seleccionado para esta asignatura, para ello revise el siguiente recurso educativo: “Variables, expresiones y sentencias” que está disponible en Lopez-Vargas, (2019).

La Figura 2 muestra en resumen las sentencias que permiten **declarar variables**, de los tres tipos de datos más comunes, tanto a nivel de mini-especificación, seudocódigo y dentro de un lenguaje de programación.

Mini-especificación	Seudocódigo	Lenguaje de programación
edad, i[1 - n]	edad : Entero	int edad
peso, d[1 - n]	peso : Real	double peso
nombre, x(50)	nombre : Cadena	String nombre

Figura 2. Declaración de variables de los tres tipos de datos bastante comunes.

Las sentencias que se utilizan para la **lectura de datos** se resumen en la Figura 3. A nivel de seudocódigo y de lenguaje de programación se necesitan de otras sentencias y elementos.

Mini-especificación	Seudocódigo	Lenguaje de programación
>>nombre	Solicitar nombre y edad	`System.out.print("Nombre:");`
>>edad	Leer nombre, edad	`nombre = lector.next();`
		`System.out.print("Edad:");`
		`edad = lector.nextInt();`

Figura 3. Sentencias para el ingreso de datos.

Los operadores aritméticos que permiten escribir algunas **expresiones aritméticas** se resumen en la Figura 4. Muchos de ellos son similares en los 3 contextos. Tanto a nivel de seudocódigo como del lenguaje de programación, no existe un operador de potencia, sino un método o función, esto se estudiará en la siguiente unidad.

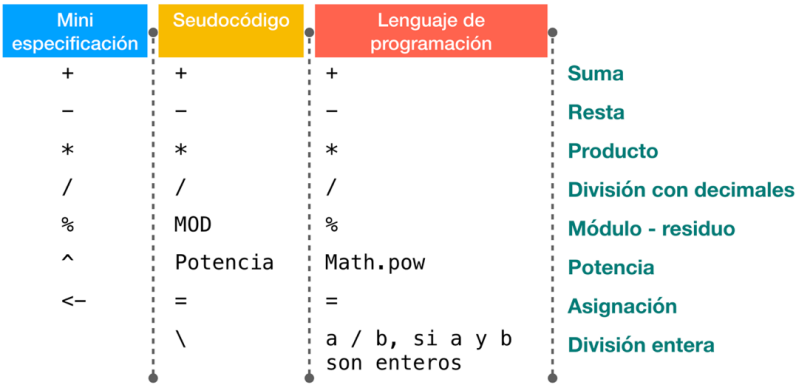


Figura 4. Operadores matemáticos

Finalmente, la escritura de datos se resume en la Figura 5, existe una gran similitud entre la mini especificación y el seudocódigo; mientras que con el lenguaje de programación como ya sucedió anteriormente se necesitan de otros elementos adicionales.

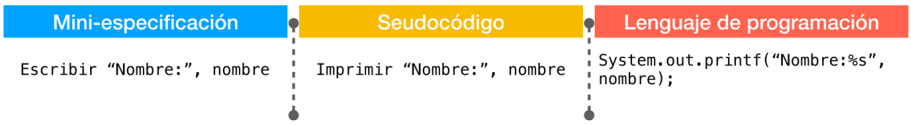


Figura 5. Escritura de datos

Si tiene alguna duda o problema con algún tema cubierto hasta el momento, no dude en manifestarla a su tutor, el le ayudará a resolverla. Recuerde que puede contactarse con él a través del servicio de mensajería interna del EVA, así como también a su correo electrónico o en el espacio de tutoría permanente.

Para finalizar le propongo que revise el siguiente problema y la implementación de una solución. El problema es encontrar el área de un triángulo. La solución a nivel de mini-especificación, seudocódigo y lenguaje de programación se muestra en la Figura 6 según los tres enfoques que se han usado anteriormente.

Mini-especificación

```
Inicio
   //declaración de variables
   AT, d[0 – n]
   b, d[0 – n]
   h, d[0 – n]
   //inicialización de variables
   AT <– 0, b <– 0, h <– 0
   //lectura de datos
   >> b
   >> h
  //proceso
   AT <– (b * h) / 2
   //escritura de datos
   << "El área es: ", AT
Fin
```

Seudocódigo

```
Algoritmo área de un triángulo
Clase Triangulo
  Método principal
     Declarar
        Variables
           area : Real
           base : Real
           altura : Real
     area = 0, base = 0 altura = 0
     Solicitar base y altura
     Leer base, altura
     area = (base * altura) / 2
     Imprimir area
  Fin Método principal
Fin Clase Triangulo
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in);
double base = 0;
double altura = 0;
var area = 0.0;
System.out.print("Ingrese la base: ");
base = l.nextDouble();
System.out.print("Ingrese el altura: ");
altura = l.nextDouble();
area = (base * altura) / 2;
System.out.printf("Área = %f\n", area);
```

Figura 6. Programas que calculan el área de un triángulo

Nunca juzgue un programa por el número de líneas que este posee, en las etapas iniciales del aprendizaje de programación, no es un buen indicador, la experiencia y la práctica le dará la posibilidad de reducir el número de sentencias que utiliza en sus programas.

De esta forma termina la primera semana de estudio de nuestra asignatura, en la próxima semana estudiará otros de los elementos que forman un programa y podrá resolver problemas un poco más difíciles.

No olvide revisar y realizar las actividades de aprendizaje recomendadas, estas le ayudarán a entender mejor, cada uno de los temas que se han desarrollado en el transcurso de esta semana.

Actividades de aprendizaje recomendadas

Responda a la siguiente Autoevaluación.

Autoevaluación 1

Ahora pondrá a prueba lo aprendido en esta unidad, para lo cual debe desarrollar la siguiente autoevaluación. Trate de responder a todas las preguntas usted mismo y luego compare sus respuestas en el solucionario que encontrará al final de este documento.

Responda a cada una de las preguntas planteadas a continuación basadas en los contenidos propuestos en esta unidad. Para responder cada pregunta es necesario que siga las instrucciones que cada una de ellas suministra.

1. En seudocódigo declare una variable para cada uno de los siguientes tipos de datos: *Cadena, Entero, Real, Booleano y Carácter.* Las variables tendrán los siguientes nombres: *nombre, edad, estatura, mayor de edad y género.*

2. ¿Cuál de los siguientes nombres se debería usar para una variable que representa el nombre de un estudiante?

   a. *nombreEstudiante: Cadena*
   b. *nombre_de_un_estudiante: Cadena*
   c. *ne: Cadena*
   d. *NOMBREESTUDIANTE: Cadena*

3. ¿Cuál de los siguientes nombres usaría para una constante que representa la cuota mínima?

   a. *cuotaMinima: Real*
   b. *cm: Real*
   c. *CUOTA_MINIMA: Real*
   d. *_cuota_minima_: Real*

4. Una variable declarada como Carácter ¿qué valor podría almacenar?

   a. Hola
   b. 123
   c. ¿?
   d. 1

5. Una variable declarada como *Entero* ¿qué valor podría almacenar?

   a. 1.1
   b. 1, 1
   c. 2
   d. 2.0

6. Una variable declarada como *Real* ¿qué valor podría almacenar?

   a. Hola
   b. 10.01
   c. =
   d. True

7. Una variable declarada como *Real* ¿qué valor podría almacenar?

   a. 1
   b. Hola
   c. Ç
   d. False

8. ¿Cuál de los tipos de datos restringe a una variable a solo dos valores?

   a. Entero
   b. Real
   c. Cadena
   d. Carácter
   e. Booleano

9. La expresión aritmética: $N = \frac{X+Y}{Y-1}$ en seudocódigo se debería escribir así:

   a. n = (x + y) / (y - 1)
   b. n = x + y / y – 1
   c. (x + y) / (y - 1) = n
   d. n = x +(y / y - 1)

10. ¿Cuál sería el valor que devolvería la siguiente expresión: 10 Mod 4?

    a. 0
    b. 1
    c. 8
    d. 2

Ir al solucionario

Ponga en práctica lo aprendido en esta unidad a través del siguiente laboratorio de programación: *López, J. (2017). Variables, entrada y salida. Recuperado de* https://goo.gl/2ozMJm.

La siguiente actividad debe enviarse al correo electrónico de su tutor. Implemente un programa Java que resuelva el siguiente problema. Una familia de tres miembros consume por día 548 litros de agua. Suponga que existe una ciudad que tiene un población de 180617 habitantes. ¿Cuántos litros de agua por día, mes y año consume esa ciudad?

Resultado de aprendizaje	Modifica y amplía programas pequeños que utilizan estándares condicionales y estructuras y funciones interactivas de control.

Contenidos, recursos y actividades de aprendizaje

La semana anterior estudió variables y operaciones elementales, esto le permitía construir programas que se asemejan a una calculadora, es decir que permiten ingresar valores que se almacenan en variables y que se usan para realizar algunos cálculos a través de un grupo de sentencias que se ejecutan una a una.

En esta semana aprenderá otros elementos básicos que forman un programa, como lo es la secuenciación y las estructuras de selección, estos elementos le permitirán construir programas con otras funcionalidades.

La base teórica para la semana la puede encontrar en: López Vargas, J. (2018) Guía Didáctica de Fundamentos de Programación. Loja –

Ecuador: Editorial UTPL, en el apartado: “UNIDAD 2. SECUENCIACIÓN Y SELECCIÓN”. Mientras que la base práctica se encuentra Lopez-Vargas, J. (2019) *Java Apuntes básicos*. Recuperado de: http://j4loxa.com/courses/java101/Java_Apuntes_Basicos.epub, en los apartados: “Funciones” y “Condicionales”.

Semana 2

Unidad 2. Secuenciación y selección

Es posible que se pregunte ¿Cuál es el orden de ejecución de las sentencias que forman un programa? Y ¿Es posible modificar ese orden? Para dar respuesta a este par de preguntas es necesario que estudie los temas que se presentan a continuación.

2.1. La secuenciación

En este apartado comprenderá sobre el orden que se sigue para ejecutar un programa. No olvide que el orden es primordial dentro de un programa, ya que los resultados de este dependen del orden.

Para comprender la fundamentación teórica es necesario que revise el siguiente el tema “2.1. La secuenciación” de López Vargas, (2018).

Al igual que las mini-especificaciones, los algoritmos en seudocódigo y los programas escritos en cualquier lenguaje de programación, emplean el mismo orden de ejecución. ¿Cuál es ese orden? Si está en la capacidad de responder a esa pregunta, avance y estudie la siguiente sección, caso contrario es necesario que vuelva a realizar la lectura recomendada.

Analice el siguiente programa escrito utilizando el lenguaje de programación Java, que tiene por objetivo calcular el promedio de ventas mensuales, usando el total de las ventas de una año. La Figura 7 muestra tres programas, en mini especificación, seudocódigo y en lenguaje de programación Java, estudie cada uno de los programas y responda a la pregunta ¿Es correcto el orden?

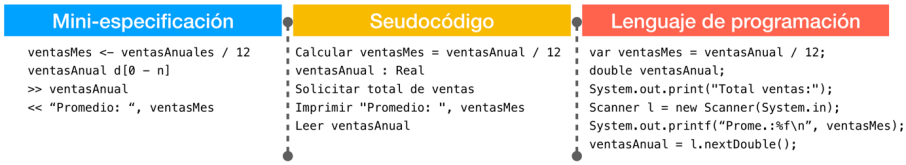


Figura 7. Programa Java para calcular el promedio de ventas anual

De seguro su respuesta es un rotundo no, el orden no es el correcto ya que el resultado que se obtiene no es el esperado. Sin embargo, el programa se ejecutará en el orden escrito, es decir, el problema no está en la forma de ejecución de los programas, está en el orden que el programador usó. A este problema se le conoce como error de lógica. ¿Puede reescribir el código en el orden correcto?

Hasta el momento conoce operaciones aritméticas básicas, pero existen otras funciones que generalmente se usan, por ejemplo, calcular la raíz cuadrada de un número o el coseno de un ángulo. Estas funciones y otras más se estudian en el siguiente tema.

2.2. Funciones matemáticas

Las funciones matemáticas son diversas y muchas de ellas ya existen implementadas en los lenguajes de programación y únicamente es necesario aprender a utilizarlas. Además de las funciones los lenguajes poseen un conjunto de constantes universales, tales como PI, que únicamente se debe utilizar.

¿Cómo se puede llamar a estas funciones matemáticas? Para dar respuesta a esta pregunta es necesario que lea el apartado "2.2 Funciones matemáticas" que se encuentra en López Vargas, (2018).

¿Es sencillo el uso de estas funciones? Posiblemente este tema sea nuevo para usted, debido a que este no se cubre en la asignatura de Algoritmos y resolución de problemas, pero como se pudo percatar su uso no es complejo.

Es necesario asociar la teoría con la práctica, es por ello que debe revisar el recurso Lopez-Vargas, (2019), el apartado "Funciones".

¿Qué le pareció el recurso? Note como el lenguaje de programación posee un grupo extenso de funciones ya predefinidas que se puede usar, aquí únicamente usaremos algunas de ellas, pero si requiere alguna función es muy probable que esta ya exista.

La Figura 8 muestra las equivalencias de las funciones matemáticas entre el seudocódigo y el lenguaje de programación Java. Es necesario mencionar que tanto a nivel de seudocódigo y Java, las funciones seno, coseno y arco tangente trabajan con valores de ángulos expresados a radianes, y es necesario realizar una transformación si los grados se encuentran expresados en grados.

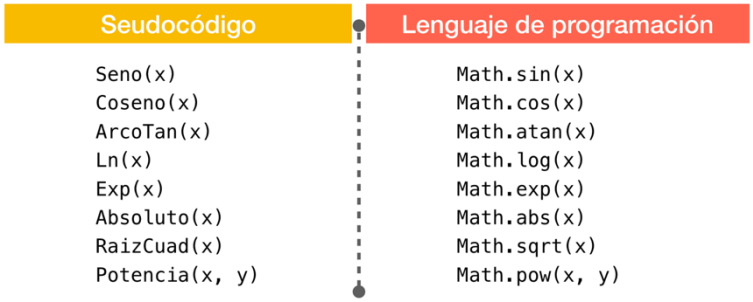


Figura 8. Tabla de equivalencia de las funciones matemáticas en seudocódigo y Java

Ahora que conoce algunas funciones matemáticas, especialmente en el lenguaje de programación Java, lo invito a que investigue ¿cómo se puede llamar a la constante PI dentro de ese lenguaje de programación? Si tuvo problemas para responder a la pregunta, la respuesta la encontrará en un programa más adelante.

El flujo normal de ejecución de un programa (secuencial) permite escribir programas cuyas sentencias se ejecutan una tras otra. La siguiente sección muestra cómo modificar ese flujo de ejecución de tal manera que seleccione qué acciones se ejecutarán y cuáles no.

2.3 Estructuras de selección

Las bases teóricas de este tema se encuentran en el apartado "2.3. Estructuras de selección" de López Vargas, (2018). ¿Recuerda las tablas de verdad para los operadores lógicos? Si aún tiene problemas, le sugiero que vuelva a realizar la lectura recomendada.

Es momento de realizar la asociación entre la teoría y la práctica, así que lo invito a leer el apartado "Condicionales" de Lopez-Vargas, (2019). ¿Qué le pareció el recurso? ¿Pudo asociar lo aprendido con lo que conocía anteriormente? Estoy seguro de que sí, de no ser así los siguientes párrafos le ayudaran.

Un elemento importante para las estructuras de selección son las operadores relacionales y lógicos. La Figura 9 se hace un resumen de los operadores. No olvide que el lenguaje de programación que usará es Java.

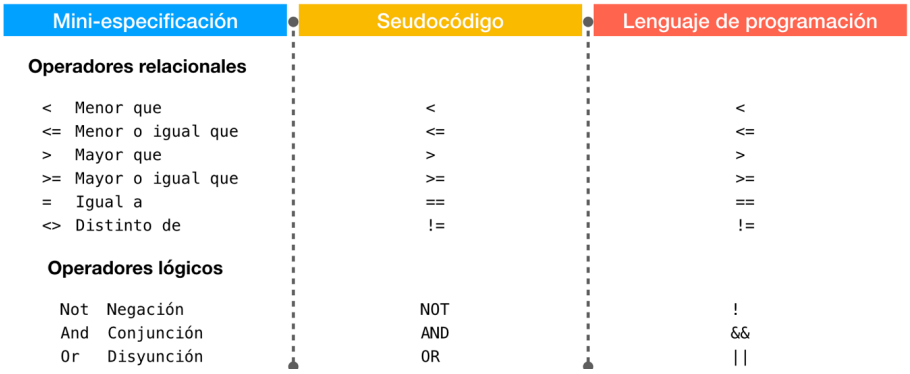


Figura 9. Comparación de los diferentes operadores relacionales y lógicos

Existen algunas diferencias entre los signos que representan cada uno de los operadores, así los signos que se usan para *"igual a"* y *"distinto de"*, en la mini especificación son diferentes a los que se usa en seudocódigo y el lenguaje de programación. Lo mismo sucede con los operadores lógicos note la diferencia marcada que existe entre el lenguaje de programación y los otros dos enfoques.

Las tres estructuras de selección se resumen en la Figura 10. Recuerde que los lenguajes de programación utilizan ciertos símbolos para agrupar sentencias y construir bloques de sentencias. En el caso del lenguaje Java, estos símbolos son las llaves ({ y }). Dentro de las estructura de selección *if…then…else* existe un bloque de sentencias que pertenecen al *if* y otras que pertenecen al *else*.

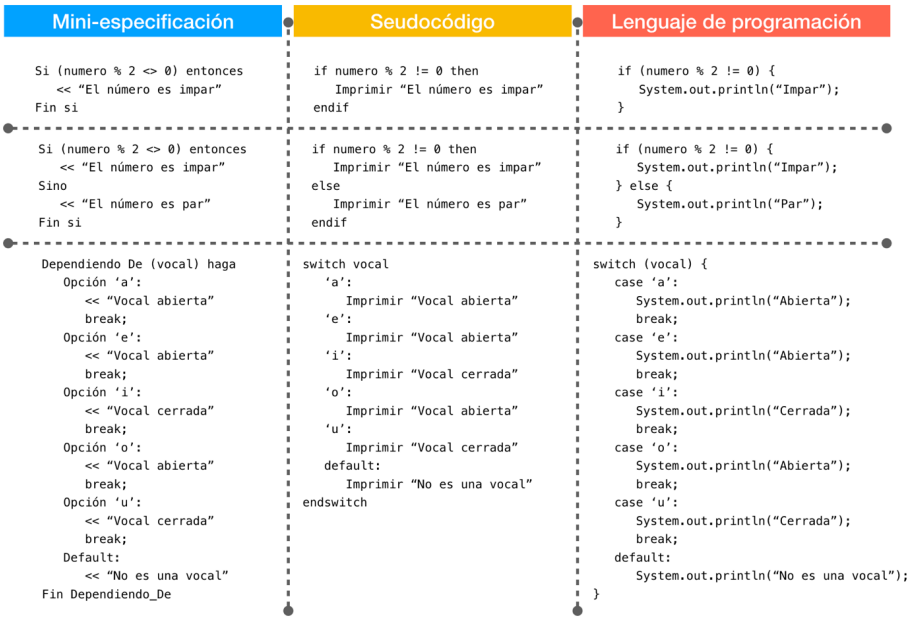


Figura 10. Resumen de 3 estructuras de selección

Para finalizar esta semana le propongo un problema sencillo. Desarrolle un programa en Java que calcule el área de un círculo. Es necesario validar que el valor que es ingresado por el usuario sea mayor que cero.

La Figura 11 muestra una posible solución al problema que se planteó anteriormente. Observe como se puede tener acceso al valor de la constante PI desde el lenguaje de programación Java.

Mini-especificación

```
Inicio
   area, d[0 – n]
   radio, d[0 – n]
   PI, d[0 – n]
   PI <– 3.1416
   >> radio
   Si(radio > 0) entonces
      area = PI * radio * radio
      Imprimir "Área=", area
   Sino
      << "Error"
   Fin si
Fin
```

Seudocódigo

```
Algoritmo Area círculo
  Clase AreaCirculo
      Método principal()
         Declarar variables
            area, radio, PI : Real
         Calcular PI = 3.1415
         Solicitar radio
         Leer radio
         if radio > 0 then
            area = PI * Pot(radio, 2)
            Imprimir "Área:", area
         else
            Imprimir "Error"
         endif
      Fin Método principal
   Fin Clase AreaCirculo
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in);
double area;
double radio;
System.out.print("Radio= ");
radio = l.nextDouble();
if (radio > 0) {
   area = Math.PI * Math.pow(radio, 2);
   System.out.printf("Área=%f\n", area);
} else {
   System.out.println("Error");
}
```

Figura 11. Programa que resuelve el problema planteado

Así termina esta segunda semana, a medida que avanza en el desarrollo de los contenidos podrá ir construyendo programas cada vez más potentes. La próxima semana inicia el estudio de otra estructura de control que le permitirá iterar (repetir) un grupo de sentencias.

Si tiene algún problema, por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o espacio de tutoría permanente.

No olvide revisar y realizar las actividades de aprendizaje tanto las recomendadas como las evaluadas, las primeras le ayudarán a entender de mejor manera cada uno de los temas que se han desarrollado en el transcurso de esta semana; mientras que las segundas le ayudan a aprobar la asignatura.

Actividades de aprendizaje recomendadas

Responda a la siguiente Autoevaluación.

Autoevaluación 2

Para determinar el nivel de comprensión de la asignatura, que hasta el momento ha alcanzado, es recomendable desarrollar la siguiente autoevaluación. Recuerde que las respuestas a todas las autoevaluaciones las puede encontrar en el solucionario que se encuentra al final de este documento.

Responda a cada una de las preguntas planteadas a continuación que se basan en los contenidos propuestos en esta unidad. En esta evaluación se plantean preguntas de programación.

1. ¿Cuál es el orden de ejecución normal de un algoritmo y de un programa?

   a. Aleatorio
   b. En paralelo
   c. Secuencial
   d. Repetitivo

2. El flujo normal de ejecución de un programa se puede modificar a través de …

   a. Estructuras de selección y repetición
   b. Arreglos
   c. Variables
   d. Constantes

3. En un algoritmo ¿quién contiene a quién?

   a. El método principal está dentro de la clase y esta a su vez se encuentra en un algoritmo.
   b. El algoritmo se encuentra dentro de la clase, la clase dentro del método principal.
   c. El método principal se encuentra dentro del algoritmo y este dentro de la clase.
   d. El algoritmo se encuentra dentro del método principal a la misma altura que la clase.

4. ¿La función Potencia también permite calcular raíces?

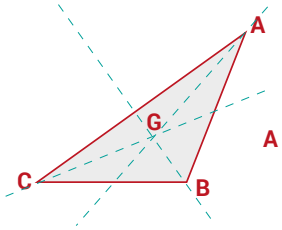
   a. Sí.
   b. No.
   c. Solo en ciertos casos.

En las siguientes preguntas es necesario que escriba programas utilizando el seudocódigo estudiado hasta el momento.

5. Escriba la estructura base de un algoritmo.

6. Elaborar un algoritmo que permita calcular la solución de una ecuación cuadrática general $ax^2 + bx + c = 0$. Se debe ingresar a, b, c. Se debe presentar el resultado de las soluciones x1 y x2.

7. Calcular la posición del baricentro. Ingresando la posición en el plano cartesiano (x, y) de los puntos A, B y C.



A $G\left(\frac{X_1+X_2+X_3}{3}, \frac{Y_1+Y_2+Y_3}{3}\right)$

8. Construya un algoritmo que al ingresar un número entero, imprima el día de la semana que le corresponde. Por ejemplo, si ingresa 1 el programa debe imprimir lunes, ingresa 2 martes, etc.

9. Elabore un algoritmo que permita calcular el índice de masa corporal IMC. Para ello debe aplicar la siguiente fórmula $IMC = \frac{Peso}{Altura^2}$, el peso debe estar expresado en kilogramos y la altura en metros. Una vez obtenido el valor de IMC, se debe usar la siguiente tabla de clasificación:

Tabla 2. Clasificación del índice de masa corporal

	Hombre	Mujer
Delgadez pronunciada	Hasta 15	Hasta 13
Delgadez	15 – 20	13 – 18
Normalidad	20 – 25	18 – 23
Gordura	25 – 30	23 – 28
Obesidad	Arriba de 30	Arriba de 28

Fuente: Rosstti (2016)

10. Elabore un algoritmo que calcule la edad de una persona. La edad debe expresarse en años, meses y días.

Ponga en práctica lo aprendido en esta unidad a través del siguiente laboratorio de programación: López, J. (2017). La secuenciación. Recuperado de https://goo.gl/bmNfLa

Ponga en práctica lo aprendido en esta unidad a través del siguiente laboratorio de programación: *López, J. (2017). La selección. Recuperado de* https://goo.gl/BcQ1KG.

Ir al solucionario

Resultado de aprendizaje

Diseña, implementa, prueba y depura un programa que utiliza cada una de las siguientes estructuras de programación: computación básica, operaciones I/O, estándares condicionales y estructuras iterativas y la definición de funciones.

Contenidos, recursos y actividades de aprendizaje

La semana anterior estudió la secuenciación y las estructuras de selección, además de otros temas tales como operadores relacionales y lógicos, estos elementos permiten construir programas que realizan cálculos y que seleccionan que acciones ejecutar.

En estas semanas aprenderá acerca de los ciclos repetitivos, que son estructuras que permiten que un bloque de sentencias se ejecute varias veces, según las necesidades del programa que está escribiendo. Una característica que comparten con las estructuras de selección es que se basan en condiciones para su funcionamiento.

También aprenderá a través del uso de series matemáticas a combinar las diferentes estructuras de control para que, a través de un trabajo coordinado, resuelvan alguna serie matemática. Este tema, sin lugar a duda será retador, pero si se hace siguiendo las instrucciones que se suministrarán es un tema relativamente sencillo.

La base teórica para las siguientes tres semanas la puede encontrar en: López Vargas, J. (2018) Guía Didáctica de Fundamentos de Programación. Loja – Ecuador: Editorial UTPL, en el apartado: "UNIDAD 3. ESTRUCTURAS DE REPETICIÓN". Mientras que la base práctica se encuentra en Lopez-Vargas, J. (2019) *Java Apuntes básicos*. Recuperado de: http://j4loxa.com/courses/java101/Java_Apuntes_Basicos.epub, en los apartados de capítulo "Iteración".

Semana 3

Unidad 3. Estructuras de repetición

Muchos de los problemas que resuelven los programas computacionales tienen tareas repetitivas, por ejemplo, calcular la nota bimestral de un grupo de estudiantes, por cada estudiante se realizan las mismas operaciones, es decir es una tarea que se repite tantas veces como estudiantes tenga el grupo.

Para trabajar con esas tareas repetitivas la mayoría de los lenguajes de programación poseen al menos una de las siguientes tres estructuras de repetición, que en las siguientes secciones y semanas aprenderá los detalles de cada una de ellas.

3.1. Ciclo repetitivo do … while

Inicie el estudio de esta nueva estructura tratando de responder a la pregunta ¿cuántas veces, como mínimo, se ejecuta el bloque de sentencias de esta estructura? Para responder a la pregunta, revise el siguiente recurso: "3.1. Ciclo repetitivo do … while" de López Vargas, (2018).

¿Encontró al recurso ilustrador? Seguro que así fue y estoy convencido que encontró la repuesta a la pregunta propuesta, sino fue así, le comento que esta estructura de repetición se repite entre 1 y n veces. Siendo n el valor máximo y 1 el mínimo, esto se debe a que la condición de la estructura se encuentra en la parte final del bloque.

Recordando los contenidos abordados en Algoritmos y resolución de problemas, la estructura *do … while* es el equivalente a la estructura lógica repetitiva *Hacer – Hasta*. Estas estructuras son similares ya que ambas estructuras tienen la condición al final, pero difieren en la formulación de la condición. No es lo mismo decir repetir hasta, que repetir mientras.

Para entender mejor la diferencia analice el ejemplo: debe ingresar el sueldo que percibe mensualmente durante un año, si se utiliza *Hacer – Hasta*, la condición sería hasta que contador sea igual a 12, mientras que si utiliza *do … while*, la condición cambia a mientras el contador es menor o igual que 12. ¿Encontró la diferencia? Estoy seguro de que así fue, caso contrario contáctese con su tutor.

A esta altura conoce los fundamentos teóricos de la estructura repetitiva y también lo relacionó con lo que ya conocía. Es hora de entrar a los detalles de un lenguaje de programación, es por ello que debe revisar el recurso: Lopez-Vargas, (2019), el apartado "Estructura do … while".

¿Pudo asociar la mini-especificación con el seudocódigo y el lenguaje de programación? La Figura 12 muestra la estructura de un ciclo *do … while* en cada uno de los tres contextos.

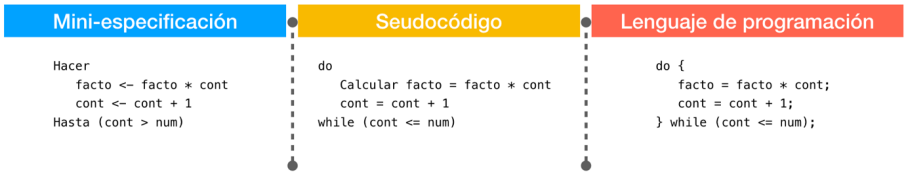


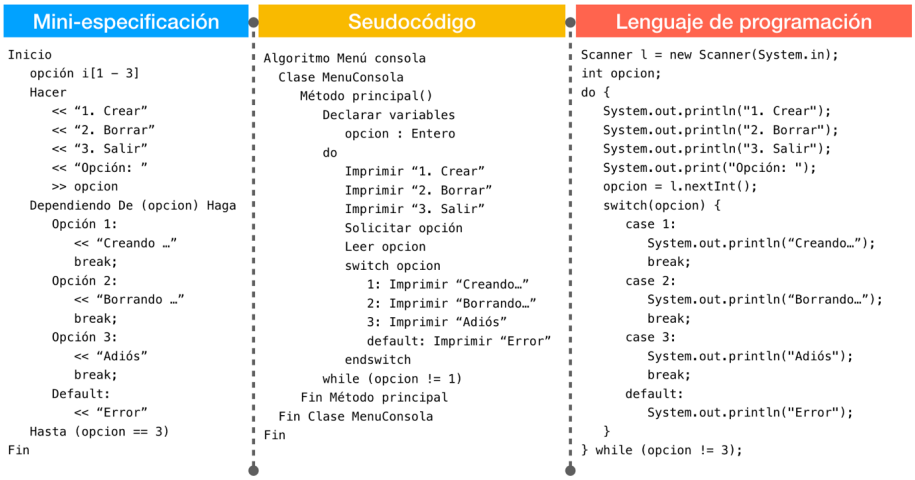
Figura 12. Estructura de repetición do ... while

Para finalizar esta semana le propongo que trate de resolver el siguiente problema. Tiene que desarrollar una aplicación en consola que presente el siguiente menú:

1. Crear
2. Borrar
3. Salir

Ahora mismo no es importante conocer que hacen las opciones crear y borrar. Lo único que nos interesa es que cuando el usuario presione el 3, se termina la ejecución del programa. Trate de resolver por su cuenta este problema, más adelante encontrará una implementación con la que comparar su respuesta.

La Figura 13 muestra una posible solución al problema propuesto. Esa solución utiliza la estructura de selección *switch* para presentar un mensaje cuando se selecciona una de las 3 opciones, mientras que presenta un mensaje de error cuando se ingresa un número que no corresponde a ninguna de las opciones. Observe como las condiciones del *do … while* utilizadas en la mini-especificación y el lenguaje de programación son diferentes como se comentó anteriormente.



Mini-especificación

```
Inicio
   opción i[1 – 3]
   Hacer
      << “1. Crear”
      << “2. Borrar”
      << “3. Salir”
      << “Opción: ”
      >> opcion
   Dependiendo De (opcion) Haga
      Opción 1:
         << “Creando …”
         break;
      Opción 2:
         << “Borrando …”
         break;
      Opción 3:
         << “Adiós”
         break;
      Default:
         << “Error”
   Hasta (opcion == 3)
Fin
```

Seudocódigo

```
Algoritmo Menú consola
  Clase MenuConsola
     Método principal()
        Declarar variables
           opcion : Entero
        do
           Imprimir “1. Crear”
           Imprimir “2. Borrar”
           Imprimir “3. Salir”
           Solicitar opción
           Leer opcion
           switch opcion
              1: Imprimir “Creando…”
              2: Imprimir “Borrando…”
              3: Imprimir “Adiós”
              default: Imprimir “Error”
           endswitch
        while (opcion != 1)
     Fin Método principal
  Fin Clase MenuConsola
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in);
int opcion;
do {
   System.out.println("1. Crear");
   System.out.println("2. Borrar");
   System.out.println("3. Salir");
   System.out.print("Opción: ");
   opcion = l.nextInt();
   switch(opcion) {
      case 1:
         System.out.println(“Creando…”);
         break;
      case 2:
         System.out.println(“Borrando…”);
         break;
      case 3:
         System.out.println("Adiós");
         break;
      default:
         System.out.println("Error");
   }
} while (opcion != 3);
```

Figura 13. Solución al problema propuesto en el apartado do ... while

Si tiene algún problema para entender este programa o para construirlo en el lenguaje de programación, por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o en el espacio de tutoría permanente.

Así termina esta tercera semana de estudio, ahora mismo ya está en la capacidad de construir programas que van más allá de simples cálculos. La próxima semana continúa estudiando una nueva estructura de control repetitiva.

Semana 4

3.2. Ciclo repetitivo for

Así como se hizo la semana anterior, en esta semana estudiará una nueva estructura repetitiva. Existen varias diferencias que las

conocerá más adelante, aquí mencionaré sólo una, la condición se evalúa antes de ingresar al bloque de sentencias a repetir.

Para continuar con los detalles de esta estructura, es necesario que revise el siguiente recurso "3.2. Ciclo repetitivo for" de López Vargas, (2018). Una vez hecha la lectura trate de responder a la pregunta ¿Cuántas veces como mínimo se ejecuta este ciclo repetitivo?

Seguro que pudo encontrar la respuesta a esta pregunta, sino es así aquí una respuesta ampliada. Considerando que una de las primeras acciones que ejecuta este ciclo repetitivo es verificar que se cumple la condición, existe la posibilidad que esa condición no llegue a cumplirse, razón por la cual nunca se ejecute el bloque de sentencias que pertenecen a la estructura repetitiva, por lo que cómo mínimo este ciclo itera 0 veces.

En la asignatura de Algoritmos y resolución de problemas estudió una estructura similar a esta que se denominaba estructura lógica repetitiva Para. De seguro pudo percatarse que tanto a nivel de seudocódigo como de mini especificación ambas estructuras tienen la misma forma ¿Lo anterior también se cumple en el lenguaje de programación?

Para dar respuesta a la pregunta anterior es necesario que revise la parte práctica, leyendo el recurso: Lopez-Vargas, (2019), concretamente el apartado "Estructura for".

Luego de revisar el recurso anterior, estoy seguro de que pudo responder a la pregunta propuesta anteriormente. Si no es así la siguiente imagen le ayudará a observar claramente que la estructura es casi la misma en los tres contextos.

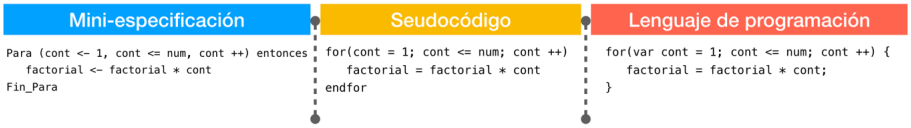


Figura 14. Estructura de un ciclo repetitivo for

Como se muestra en la Figura 14, en el lenguaje de programación Java se puede declarar la variable de control, no sólo darle un valor inicial como se hace en la mini especificación y el seudocódigo. Un última diferencia es que en la mini especificación se utiliza la coma (,) para separar cada una de las partes que forman el ciclo, mientras que a nivel de seudocódigo y lenguaje de programación se utiliza punto y coma (;).

Antes de terminar con el desarrollo de los contenidos previstos en esta semana, es necesario resolver un problema con lo aprendido hasta el momento. El problema consiste en verificar si un número es primo o no. Un número primo es un número natural mayor que 1 que únicamente tiene dos divisores distintos: él mismo y el 1.

En la Figura 15 se muestran 3 programas que resuelven el problema propuesto anteriormente, para la construcción de los programas se utilizó mini especificación, seudocódigo y Java. Cada uno de esos programas son equivalentes.

Mini-especificación

```
Inicio
   nro, i[1 – n]
   div, i[2 – n]
   contDiv, i[0 – n]
   contDiv <– 0
   << “Ingrese un número”
   >> nro
   Para (div = 2, div < nro, div ++) entonces
      Si (nro % div = 0) entonces
        contDiv = contDiv + 1
      Fin si
   Fin_Para
   Si (contDiv = 0) entonces
      << “Es primo”
   Sino
      << “No es primo”
   Fin si
Fin
```

Seudocódigo

```
Algoritmo Determina primo
  Clase Primo
    Método principal()
      Declarar variables
        nro, div, contDiv: Entero
      Calcular contDiv = 0
      for(div = 2; div < nro; div ++)
        if nro MOD div == 0 then
          contDiv = contDiv + 1
        endif
      endfor
      if contDiv == 0 then
        Imprimir “Es primo”
      else
        Imprimir “No es primo”
      endif
    Fin Método principal
  Fin Clase Primo
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in);
int nro;
int contDiv;
contDiv = 0;
nro = l.nextInt();
for(var div = 2; div < nro; div ++)
   if(nro % div == 0) {
      contDiv = contDiv + 1;
   }
if(contDiv == 0) {
   System.out.println("Es primo");
} else {
   System.out.println("No es primo");
}
```

Figura 15. Programas que resuelven el problema propuesto

Si analiza a detalle, los programas anteriores no funcionan correctamente para todos los casos. ¿Puede ver el error? Estoy seguro de que así fue, caso contrario lo invito a que haga una prueba de escritorio de cada programa e ingreso el número -1 (menos uno). ¿Qué resultado obtuvo?

Si tiene algún problema para entender este programa o para construirlo en el lenguaje de programación o no encuentra el error en el mismo, por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o en el espacio de tutoría permanente.

De esta forma concluye la cuarta semana de estudio. Ya conoce 2 estructuras repetitivas, pero aún falta conocer una más. La próxima semana estudiará la última estructura de repetición.

Semana 5

3.3. Ciclo repetitivo while

Esta semana aprenderá a utilizar la última estructura repetitiva denominada *while*. Su funcionamiento es una mezcla entre las dos estructuras anteriores, es decir, se asemeja al ciclo *for* debido a que primero evalúa la condición y luego ejecuta su bloque de sentencias, mientras que se asemeja al *do … while* ya que la forma de construir las condiciones es la misma.

Para conocer los detalles de esta nueva estructura de repetición revise el siguiente recurso: “3.3. Ciclo repetitivo while” de López Vargas, (2018). Una vez que concluya con la revisión del recurso trate de responder a la pregunta ¿con qué estructura lógica repetitiva que estudió en Algoritmos y resolución de problemas se asemeja el *while*?

Seguro que su respuesta es con la estructura lógica repetitiva *Mientras que – Hacer*. Ambas estructuras se ejecutan como entre 0 y n veces ya que tienen la condición al inicio, por estas características anteriormente se dijo que se parece al ciclo repetitivo *for*.

De seguro se está preguntando ¿cómo se escribe un ciclo repetitivo while en el lenguaje de programación Java? Para que pueda responder a esta pregunta lee el recurso: "Estructura while" de Lopez-Vargas, (2019).

Ahora ya conoce la forma de esta estructura de control a nivel de mini especificación, seudocódigo y el lenguaje de programación Java. La Figura 16 resume la forma de esta estructura en esos tres contextos.

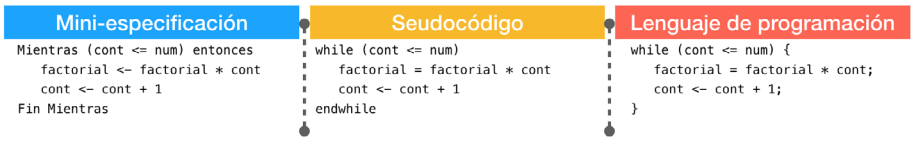


Figura 16. Estructura repetitiva while

Note como a diferencia de la estructura lógica repetitiva *Hacer – Hasta*, en los tres enfoques la condición se escribe de igual manera. Recuerde que es diferente escribir condiciones "hacer hasta" que "hacer mientras". Si no recuerda este tema revise el tema 3.1. Ciclo repetitivo do … while de la semana 3.

Es momento de plantear un problema y que trate de resolverlo usted mismo, luego puede ver una solución al mismo. El problema dice lo siguiente: Se necesita calcular el factorial de un número. El factorial de un número *n*, que es entero positivo, se define como el producto de todos los números enteros positivos desde 1 hasta n. Para resolver este problema debe desarrollar un programa utilizando el lenguaje de programación Java.

De seguro resolvió el problema y creó su propio programa. La Figura 17 muestra la implementación de una posible solución a ese problema, como se ha venido haciendo se usa mini especificación, seudocódigo y Java.

Mini-especificación

```
Inicio
   cont, i[0 – n]
   num, i [0 – n]
   factorial, i[0 – n]
   cont <– 1, factorial <– 1
   << “Ingrese n”
   >> num
   Si (num < 0) entonces
      << “Error”
   Sino
      Mientras (cont <= num) entonces
         factorial <– factorial * cont
         cont <– cont + 1
      Fin_Mientras
      << “Factorial=“, factorial
   Fin si
Fin
```

Seudocódigo

```
Algoritmo Factorial
  Clase Factorial
    Método principal()
      Declarar variables
        cont, num, factorial : Entero
      Calcular cont = 1, factorial = 1
      Solicitar un número
      Leer num
      if num < 0 then
        Imprimir “Error”
      else
        while(cont <= num)
          factorial = factorial * cont
          cont = cont + 1
        endwhile
        Imprimir “Factorial=“, factorial
      endif
    Fin Método principal
  Fin Clase Factorial
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in);
var cont = 1,
var factorial = 1;
int num;
System.out.print("Ingrese un número: ");
num = l.nextInt();
if (num < 0) {
   System.out.println(“Error”)
} else {
   while(cont <= num) {
      factorial = factorial * cont;
      cont = cont + 1;
   }
   System.out.printf(“Fact.=%d\n", factorial);
}
```

Figura 17. Cálculo del factorial de un número usando un while

Los tres programas anteriores son equivalentes, existen algunas diferencias sintácticas y como se inicializan las variables, especialmente en el lenguaje de programación ya que se utiliza inferencia de datos para las variables *cont* y *factorial*, es decir, no se especifica el tipo de dato este se deduce por el valor inicial que recibe.

Si tuvo algún problema para comprender este programa o para construirlo, por favor comuníquese con su tutor, él le ayudará a resolverlos. Los medios de comunicación que dispone son mensajería interna del entorno, correo electrónico o en el espacio de tutoría permanente.

En el siguiente apartado aprenderá un conjunto de criterios que le permitirán seleccionar una de las tres estructuras repetitivas que ya conoce.

3.4. ¿Cuándo usar cada ciclo repetitivo?

Ahora conoce tres estructuras repetitivas que son equivalentes y en la mayoría de los casos intercambiables entre si, es decir se puede usar cualquiera. Es posible entonces que se pregunte ¿cómo seleccionar una estructura repetitiva? La respuesta a esa pregunta se encuentra en: López Vargas, (2018) bajo el título "3.4 ¿Cuándo usar cada ciclo repetitivo?"

Luego de la lectura anterior es necesario aclarar el punto 2 que se resumen así: usar *while* cuando se debe modificar la variable de control cuando únicamente se cumplen algunas condiciones. Con ese objetivo, es necesario que analice el siguiente programa. Este programa escrito en Java genera aleatoriamente números enteros comprendidos entre 1 y 100, el programa termina cuando se han generado 3 números pares.

La Figura 18 muestra la implementación del programa y señala varios elementos que se explican a continuación.

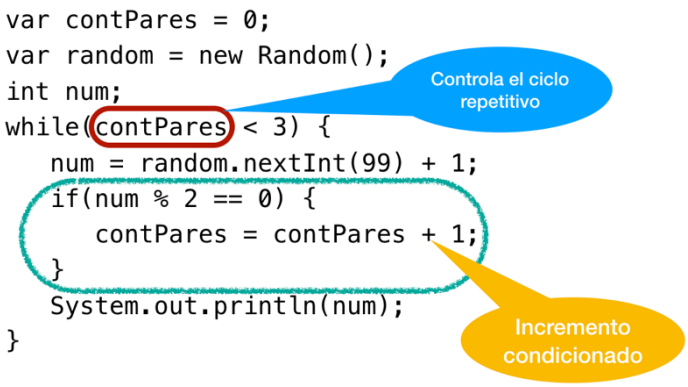


Figura 18. Programa Java que resuelve el problema propuesto anteriormente

La variable que contrala un ciclo repetitivo es aquella que es parte de su condición, en el programa es la variable *contPares*, esa variable cambia de valor cuando el número generado es par, es por ello que

su incremento se encuentra condicionado, es decir, se debe cumplir una condición para incrementar su valor.

Se cierra esta quinta semana de estudio ¿cómo ha asimilado cada uno de los temas? Estoy seguro de que ya tiene dominio de muchos de ellos y de algunos otros aún no. La próxima semana podrá reforzar cada uno de los temas ya que trabajará con problemas de mayor complejidad que le ayudarán a crear problemas más complejos.

Semana 6

3.5. Series numéricas

Hasta el momento los problemas que ha estudiado, y de seguro implementado, utilizan pocos elementos de los que ha aprendido, es por ello que en esta semana aprenderá a resolver problemas de mayor complejidad. Estos problemas son extraídos del mundo de las matemáticas y generalmente se denominan sucesiones numéricas.

El detalle del tema lo puede encontrar en: López Vargas, (2019) tema: “3.5 Series numéricas”. Revise el tema propuesto con especial énfasis en el ejemplo que el se encuentra en dicha sección.

¿Pudo comprender las series numéricas? De seguro su respuesta es un sí algo inseguro. Para eliminar la inseguridad lo invito a revisar los siguiente párrafos.

Las series numéricas que usará en esta asignatura se pueden clasificar en dos grupos, así:

- Series en donde existe una fórmula para calcular cada término de esta. Por ejemplo: Fibonacci. La fórmula que se aplica es: a partir del tercer elemento estos se generan sumando los dos anteriores. Para trabajar con este tipo de series se recomienda usar un ciclo *for*. Analice el siguiente programa, ver Figura 19, que suma los 10 primeros términos de la sucesión de Fibonacci. [FCE: Insertar la imagen: estructurasrepeticion8]

Mini-especificación

```
Inicio
  primero, i[0 – n]
  segundo, i [0 – n]
  fibo, i[0 – n]
  suma, i[0 – n]
  cont, i[0 – n]
  primero <– 0, segundo <– 1, suma <– 1
  Para(cont <– 2, cont< 8, cont++) entonces
   fibo = primero + segundo
   primero = segundo
   segundo = fibo
   suma = suma + fibo
  Fin_Para
  <<“Suma=”, suma
Fin
```

Seudocódigo

```
Algoritmo Suma 10 números Fibonacci
  Clase SumaFibo
    Método principal()
      Declarar variables
        primero, segundo, fibo : Entero
        suma, cont : Entero
      Calcular primero = 0, segundo = 1
      Calcular suma = 1
      for(cont = 0; cont < 8; cont ++)
         fibo = primero + segundo
         primero = segundo
         segundo = fibo
         suma = suma + fibo
      endfor
      Imprimir “Suma=”, suma
   Fin Método principal
  Fin Clase SumaFibo
Fin
```

Lenguaje de programación

```
var primero = 0
var segundo = 1
int fibo
var suma = 1
for(var cont = 0; cont < 8; cont ++) {
   fibo = primero + segundo;
   primero = segundo;
   segundo = fibo;
   suma = suma + fibo;
}
System.out.printf(“Suma=%d\n”, suma);
```

Figura 19. Programa que suma los 10 primeros términos de Fibonacci

La fórmula es parte del ciclo *for*, además se hace un intercambio de valores con el fin de preparar las variables para producir los próximos valores. Ahora le propongo dos preguntas ¿Porqué la variable suma tiene un valor inicial de uno? Y ¿Porqué la condición del ciclo *for* va hasta 8 cuando se necesitan 10 términos? Las respuestas a ambas preguntas tienen su origen en que los dos primeros términos de la sucesión no se generan, se generan del tercero en adelante.

Series en donde los términos que la forman deben cumplir un grupo de condiciones. Por ejemplo: números primos. Cada término de esa serie debe tener sólo dos divisores. Aquí se recomienda usar un ciclo *while*. Un ejemplo de la solución se presenta en la Figura 20.

Mini-especificación

```
Inicio
  cont, i[0 – n]
  suma, i[0 – n]
  num, i[0 – n]
  div, i[0 – n]
  esPrimo, l
  num <– 1, suma <– 0, cont <– 0
  Mientras (cont < 10)
    num <– num + 1
    esPrimo <– True
    Para(div <– 2, div < num, div ++) entonces
      Si (num % div = 0) Entonces
        esPrimo <– False
      Fin Si
    Fin_Para
    Si (esPrimo = True) Entonces
      suma <– suma + num
      cont <– cont + 1
    Fin Si
  Fin_Mientras
  << "Suma=", suma
Fin
```

Seudocódigo

```
Algoritmo Suma 10 números primos
  Clase SumaPrimo
    Método principal()
      Declarar variables
        cont, suma, num, div : Entero
        esPrimo : Booleano
      Calcular num = 1, suma = 0
      Calcular cont = 0
      while (cont < 10)
        num = num + 1
        esPrimo = True
        for(div = 2; div < num, div ++)
          if num MOD div == 0 then
            esPrimo = False
          endif
        endfor
        if esPrimo == True then
          suma = suma + num
          cont = cont + 1
        endif
      endwhile
      Imprimir "Suma=", suma
    Fin Método principal
  Fin Clase SumaPrimo
Fin
```

Lenguaje de programación

```
var cont = 0;
var num = 1;
var suma = 0;
boolean esPrimo;
while(cont < 10) {
   num = num + 1;
   esPrimo = true;
   for(var div = 2; div < num; div ++) {
      if(num % div == 0) {
         esPrimo = false;
      }
   }
   if(esPrimo == true) {
      suma = suma + num;
      cont = cont + 1;
   }
}
System.out.printf("Suma = %d\n", suma);
```

Figura 20. Programa que suma los 10 primeros primos

La variable *num* tiene que tener únicamente dos divisores para ser parte de la serie y como lo vio anteriormente en este caso es una suma e incremento condicionado, es por esto que se recomienda el uso del ciclo *while*.

Es posible que ahora mismo se esté preguntando ¿cómo trabajar con series que combinen ambos tipos? En este caso, lo primero es encontrar un término que cumple las condiciones y luego emplear la fórmula para calcular término de otro tipo de serie. Para comprender mejor el tema, se sugiere que revise el ejercicio que se mencionó al inicio de la semana.

Para finalizar esta semana resuelva la siguiente serie numérica:

$$S = -7^0 + 11^1 + 13^1 - 17^2 - 19^3 - 23^5 + \ldots$$

Figura 21. Serie numérica a resolver

Como puede observar, se trata de una suma de potencias, en donde existe una combinación de números primos, que se ubican en la base de cada potencia y Fibonacci como exponentes. Una solución se muestra a continuación, ver la Figura 22. La sección A muestra la declaración de variables y el ingreso de datos, mientras que la B muestra las sentencias que resuelven el problema. [FCE: Insertar la imagen: estructurasrepeticion11]

A

```
Scanner l = new Scanner(System.in);
var suma = 0.0;
var num = 6;
int fibo;
var primero = 0;
var segundo = 1;
boolean esPrimo;
var cont = 0;
var signo = -1;
var contSignos = 0;
var cambiaSigno = 1;
int limite;
System.out.print("Ingrese límite:");
limite = l.nextInt();
```

B

```
while(cont < limite) {
   num = num + 1;
   esPrimo = true;
   for(var div = 2; div < num; div ++) {
      if(num % div == 0) {
         esPrimo = false;
      }
   }
   if(esPrimo == true) {
      if(cont == 0 || cont == 1) {
         fibo = cont;
      } else {
         fibo = primero + segundo;
         primero = segundo;
         segundo = fibo;
      }
      System.out.printf("(%d)%d^%d\n", signo, num, fibo);
      suma = suma + Math.pow(num, fibo) * signo;
      cont = cont + 1;
      contSignos = contSignos + 1;
      if(contSignos == cambiaSigno) {
         signo = signo * -1;
         cambiaSigno = cambiaSigno + 1;
         contSignos = 0;
      }
   }
}
System.out.printf("Suma=%f\n", suma);
```

Figura 22. Programa que resuelve la serie propuesta

Algunos comentarios de ese programa, primero se encuentra el número primo, luego se aplica la fórmula para encontrar Fibonacci. En este problema se debe trabajar con los dos primeros términos de Fibonacci, recuerde que estos no se generan, es por ello que se debe controlar en qué término se encuentra, si es el primero o el segundo (*if(cont == 0 || cont == 1) { …*), del tercero en adelante se aplique la fórmula que estudió anteriormente.

De esta forma concluye el desarrollo de contenidos de este Primer Bimestre, en las siguientes semanas debe dedicarse al desarrollo de la tarea y a prepararse para la evaluación presencial.

No olvide revisar y realizar las actividades de aprendizaje recomendadas, estas le ayudarán a entender de mejor manera cada uno de los temas que se han desarrollado en el transcurso de estas semanas.

Actividades de aprendizaje recomendadas

Responda a la siguiente Autoevaluación.

Autoevaluación 3

Es el momento de comprobar lo aprendido en esta unidad, para lo cual debe desarrollar la siguiente autoevaluación. Trate de resolverla usted mismo y compare su respuesta con el solucionario que encontrará al final de este documento.

Responda a cada una de las preguntas planteadas, a continuación que se basan en los contenidos propuestos en esta unidad. En esta evaluación se plantean preguntas de programación.

1. En un ciclo repetitivo *do…while* la condición se ubica.

    a. Al inicio del ciclo
    b. Al final del ciclo
    c. En la mitad
    d. No tiene condición

2. En un ciclo *for* la condición puede ser

    a. Únicamente compuesta
    b. Únicamente simple
    c. Compuesta o simple
    d. No posee condición

3. En un ciclo repetitivo *while*, la variable que controla el ciclo repetitivo es aquella que:

    a. Se asigna un valor inicial dentro del ciclo
    b. Forma parte de la condición del ciclo
    c. Es constante
    d. Es modificada fuera del ciclo repetitivo.

4. El ciclo repetitivo que se ejecuta al menos una vez es el ciclo.

   a. *for*
   b. *do…while*
   c. *while*
   d. *switch*

5. En cuál de los ciclos repetitivos no se recomienda cambiar el valor de la variable de control dentro de las acciones/ sentencias que forman ese ciclo.

   a. *for*
   b. *do…while*
   c. *while*
   d. *if-then-else*

En las siguientes preguntas es necesario que escriba programas utilizando el seudocódigo estudiado hasta el momento.

6. Elabore un algoritmo que haga la suma de todos los números enteros positivos que son menores o iguales a un valor ingresado por el usuario.

7. Elabore un algoritmo que permita encontrar los 3 primeros números perfectos. Un número es perfecto cuando la suma de sus divisores enteros positivos menores es igual al mismo número. Ejemplo: el número 6 es perfecto, porque sus divisores son (enteros positivos menores): 1, 2 y 3; y sumamos: 1 + 2 +3 = 6

8. Elabore un algoritmo que calcule y presente la sumatoria de los n primeros términos de una serie formada por números primos (un número primo es el que es divisible únicamente para sí mismo y la unidad). La representación de la serie sería:

$$S = 3 + 5 + 7 + 11 + 13 + 17 + 19 + 23 + \cdots$$

9. Elabore un algoritmo que encuentra la suma de las n primeros términos que pertenecen a la serie de Fibonacci. Una representación de la serie es:

$$S = 0 + 1 + 1 + 2 + 3 + 5 + 8 + 23 + 21 + \cdots$$

10. Elabore un algoritmo y programa en Java que encuentre la sumatoria de los n primeros términos de la siguiente serie:

$$S = 3^0 + 5^1 + 7^1 + 11^2 + 13^3 + 17^5 + 19^8 + \cdots$$

Ir al solucionario

Revise el siguiente vídeo que le ayudará comprender el funcionamiento del ciclo repetitivo do … while. Jumbo, L. (2016) Do... While. Recuperado de https://youtu.be/89AU9kEzEUM.

Ponga en práctica lo aprendido en esta unidad a través del siguiente laboratorio de programación: López, J. (2017). Ciclo do...while. Recuperado de https://goo.gl/T31GCa.

Revise el siguiente vídeo que le ayudará comprender el funcionamiento del ciclo repetitivo for. Microvídeos UTPL. (2016) Bucle For. Recuperado de https://youtu.be/8G1IlOwVvL4.

Ponga en práctica lo aprendido en esta unidad a través del siguiente laboratorio de programación: López, J. (2017). Ciclo for. Recuperado de https://goo.gl/ae4awR.

Revise el siguiente vídeo que le ayudará comprender el funcionamiento del ciclo repetitivo While. Quiñones, S. (2016) Estructura repetitiva While. Recuperado de https://youtu.be/N8HQRFN9i-g.

Ponga en práctica lo aprendido en esta unidad a través del siguiente laboratorio de programación: López, J. (2017). Ciclo while. Recuperado de https://goo.gl/XWdKhP.

Para comprender de mejor forma las estructuras de repetición revise el sitio Web: El blog del Profe Alex Recuperado de: https://profe-alexz.blogspot.com.

Revise el siguiente vídeo que le ayudará comprender las series numéricas. Academia Internet (2017) Sucesiones numéricas, aritméticas, geométricas, primos, Fibonacci, potencias. Recuperado de https://youtu.be/bqg-15oePxs.

Ponga en práctica lo aprendido en esta unidad a través del siguiente laboratorio de programación: López, J. (2017). Ciclo while. Recuperado de https://goo.gl/oXP3bB.

Actividades finales del bimestre

Semana 7

En esta semana es recomendable que vuelva a revisar los contenidos que se desarrollaron en las semanas 1, 2 y 3.

No olvide de practicar creando varios programas utilizando especialmente el lenguaje de programación Java.

Si tiene algún problema, por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o en el espacio de tutoría permanente.

Semana 8

Para esta última semana de preparación se recomienda que haga una revisión de los contenidos que se cubrieron en las semanas 4, 5 y 6.

No olvide de practicar creando varios programas utilizando especialmente el lenguaje de programación Java.

Si tiene algún problema, por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o en el espacio de tutoría permanente.

Segundo bimestre

Resultado de aprendizaje

Escribe programas que utilizan cada una de las siguientes estructuras de datos: arreglos.

Contenidos, recursos y actividades de aprendizaje

El bimestre anterior conoció los elementos fundamentales que permiten crear muchos programas, pero existen otros conceptos que son de uso masivo en programación, es por ello que debe estudiar un nuevo grupo de elementos que van a potenciar aún más los programas que construye.

Uno de esos nuevos elementos son las estructuras de datos. Si bien es un tema extenso, ya que existen muchas de esas estructuras, empezará su estudio con una estructura sencilla que se denomina arreglos. Aprenderá las características de estos, los tipos de arreglos que existen y las operaciones básicas que se pueden ejecutar con esta estructura de datos.

Así como se hizo anteriormente la base teórica para estas semanas la encontrará en: López Vargas, J. (2018) Guía Didáctica de Fundamentos de Programación. Loja – Ecuador: Editorial UTPL, en el apartado: "UNIDAD 4. ARREGLOS". Mientras que la base práctica se encuentra en Lopez-Vargas, J. (2019) *Java Apuntes básicos*. Recuperado de: http://j4loxa.com/courses/java101/Java_Apuntes_Basicos.epub, en los apartados: "Arreglos".

Semana 9

Unidad 4. Arreglos

Las estructuras de datos de forma general se clasifican en estáticas y dinámicas cuando se considera su capacidad de crecer o de aumentar de tamaño. Las primeras no pueden crecer, mientras que las segundas sí. En esta semana estudiará una de las primeras estructuras estáticas, concretamente los arreglos unidimensionales.

4.1. Arreglos unidimensionales

Como ya se mencionó anteriormente los arreglos son estructuras de datos estáticas, pero esta no es la única característica ¿Qué otras características tienen los arreglos? Para responder a esta pregunta debe revisar el siguiente recurso: “4.1. Arreglos Unidimensionales” de López Vargas, (2018).

¿Comprendió las características de los arreglos unidimensionales? Al trabajar con arreglos, los errores más comunes es trabajar fuera de los límites de este, por ejemplo, en el lenguaje de programación Java, este error se identifica con una excepción que tiene el siguiente nombre *ArrayIndexOutOfBoundsException*.

Antes de continuar con el estudio de esta estructura de datos es necesario que recuerde que este tema ya lo estudió en la asignatura de Algoritmos y resolución de problemas bajo el mismo nombre.

Una vez estudiados los fundamentos teóricos, es necesario pasar al tema práctico. Para ello revise el siguiente recurso: Lopez-Vargas, (2019), realizar una revisión a conciencia del tema "Arreglos".

Continúe con el desarrollo de los contenidos de esta semana y estudie algunas operaciones básicas para el trabajo con los arreglos. Inicie con la declaración de un arreglo, así como se hizo anteriormente se presentará la mini especificación, el seudocódigo y el lenguaje de programación Java, la Figura 23 muestra las sentencias que se utilizan para esta tarea.

Figura 23. Declaración de un arreglo

En los tres contextos se hace la declaración de una variable que representa a un arreglo que tiene un tamaño de 17. En el leguaje de programación se pueden realizar diferentes formas de creación. La primera, utiliza inferencia de tipos de dato para determinar el tipo de dato de la variable arreglo. La segunda es posiblemente la forma tradicional de declarar arreglos. En la tercera línea se hace la declaración e inicialización de la variable. Antes de continuar le propongo que responda a la pregunta ¿Cuál es la longitud del arreglo *text*? Estoy seguro de que su respuesta fue tres.

Luego de crear el arreglo generalmente se itera sobre cada una de las posiciones, ya sea para ingresar valores o recuperar los existentes. Esta tarea se realiza con una estructura repetitiva, generalmente un ciclo *for*. Revise la Figura 24 ya que resume como realizar el ingreso de datos y luego la presentación de los valores que contiene el arreglo.

Mini-especificación

```
Inicio
  arreglo[7], i[0 – n]
  i, i[0–7]
  Para(i <- 0, i < 7, i ++) Entonces
    << “Solicitar valor”
    >> arreglo[i]
  Fin_Para
  Para(i <- 0, i < 7, i ++) Entonces
    << “Valor=”, arreglo[i]
  Fin_Para
Fin
```

Seudocódigo

```
Algoritmo Ingreso/mostrar arreglo
  Clase Arreglo
    Método principal()
      Declarar variables
        arreglo : Arreglo[7] Entero
        i : Entero
      for(i = 0; i < 7; i ++)
        Solicitar valor
        Leer arreglo[i]
      endfor
      for(i = 0; i < 7; i ++)
        Imprimir arreglo[i]
      endfor
    Fin Método principal
  Fin Clase Arreglo
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in)
var arreglo = new int[7];

for(var i = 0; i < arreglo.length; i++) {
   System.out.print(“Valor: ”);
   arreglo[i] = l.nextInt();
}

for(var i = 0; i < arreglo.length; i++) {
   System.out.printf(“%d\n”, arreglo[i]);
}

for(var valor : arreglo) {
   System.out.printf(“%d\n”, valor);
}
```

Figura 24. Ingreso y presentación de datos en un arreglo

En el lenguaje de programación Java para la presentación de los valores de un arreglo se muestran dos opciones, la primera un *for* tradicional, mientras la segunda utiliza lo que se denomina un *forEach* o un *for* simplificado. En el primer ciclo repetitivo, se consulta el tamaño del arreglo a través de la propiedad *length*. El segundo ciclo repetitivo es útil cuando no es necesario el índice de las posiciones del arreglo.

Antes de finalizar esta semana es conveniente que ponga en práctica lo aprendido resolviendo un problema en donde utilice arreglos unidimensionales. El problema que tiene que resolver es el siguiente: en un paralelo de Fundamentos de programación existen 10 estudiantes que han rendido una evaluación presencial y necesita obtener el promedio. Intente resolver el problema propuesto, más adelante encontrará una imagen con una posible solución.

Mini-especificación

```
Inicio
  notas[10], d[0 – n]
  i, i[0 – 10]
  suma, d[0 – n]
  promedio, d[0 – n]
  suma <– 0
  Para(i <– 0, i < 10, i ++) Entonces
    << “Solicitar nota”
    >> notas[i]
  Fin_Para
  Para(i <– 0, i < 10, i ++) Entonces
    suma <– suma + notas[i]
  Fin_Para
  promedio <– suma / 10
  << “Promedio=”, promedio
Fin
```

Seudocódigo

```
Algoritmo Promedio notas
  Clase PromedioNotas
    Método principal()
      Declarar variables
        notas : Arreglo[10] Real
        i : Entero
        suma : Real
        promedio : Real
      Calcular suma = 0
      for(i = 0; i < 10; i ++)
        Solicitar nota
        Leer notas[i]
      endfor
      for(i = 0; i < 10; i ++)
        suma = suma + notas[i]
      endfor
      Calcular promedio = suma / 10
      Imprimir “Promedio=“, promedio
    Fin Método principal
  Fin Clase PromedioNotas
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in)
var notas = new double[10];
var suma = 0.0;
double promedio;

for(var i = 0; i < notas.length; i++) {
   System.out.print(“Nota: ”);
   notas[i] = l.nextDouble();
}

for(var nota : notas) {
   suma = suma + nota;
}
promedio = suma / notas.length;
System.out.printf(“Promedio=”, promedio);
```

Figura 25. Programa que calcula el promedio de las notas

En el programa desarrollado en Java se puede utilizar la propiedad *length* para calcular el promedio, esta característica permite que se cambie el tamaño del arreglo y no se necesite modificar las líneas de código que lo usen. Lamentablemente esto no se puede aplicar en la mini especificación ni el seudocódigo, aunque se podría mejorar su código si se declara una variable con el tamaño del arreglo. ¿Entiende como realizar este cambio?

Si tiene algún problema para implementar el cambio propuesto o con algún contenido por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o el espacio de tutoría permanente.

La próxima semana continuará estudiando los arreglos, pero en este caso hará énfasis en los arreglos de más de una dimensión.

Semana 10

4.2. Arreglos bidimensionales

Los arreglos bidimensionales son otras de las estructuras que se pueden considerar básicas, comparten varias características con los arreglos. ¿Cuál es la diferencia entre arreglos y matrices? Para responder a esta pregunta revise el siguiente recurso: "4.2. Arreglos bidimensionales o matrices" de López Vargas, (2018).

De seguro esta otra estructura de datos no le resultó nueva ya que la estudió en Algoritmos y resolución de problemas. Ahora ya tiene clara la diferencia entre ambas estructuras de datos, recuerde que tradicionalmente las matrices se recorren primero por fila y por cada una de ellas, las columnas.

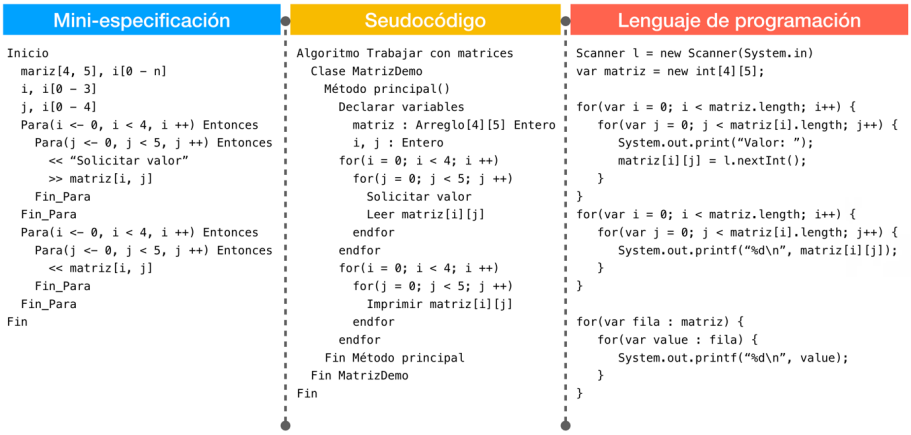
Es momento de ir a la práctica, para ello es necesario que revise el tema: "Arreglos bidimensionales" del recurso Lopez-Vargas, (2019), que le mostrará como se implementan las matrices en el lenguaje de programación Java.

¿Qué le pareció el recurso? De seguro lo pudo aprovechar al máximo y le ayudó a conocer cómo se crean e iteran matrices en un lenguaje de programación de tercer nivel. A manera de síntesis revise la Figura 26 que muestra como se puede crear una matriz en mini especificación, seudocódigo y Java.

Figura 26. Creación de una matriz

Dentro del lenguaje de programación Java se puede utilizar inferencia de tipos para crear matrices o se puede explicitar el tipo de dato. También se puede crear y dar un valor inicial a una matriz.

¿Cómo se puede iterar en una matriz? La respuesta a esta pregunta se resume en la Figura 27.



Mini-especificación

```
Inicio
  mariz[4, 5], i[0 – n]
  i, i[0 – 3]
  j, i[0 – 4]
  Para(i <– 0, i < 4, i ++) Entonces
    Para(j <– 0, j < 5, j ++) Entonces
      << "Solicitar valor"
      >> matriz[i, j]
    Fin_Para
  Fin_Para
  Para(i <– 0, i < 4, i ++) Entonces
    Para(j <– 0, j < 5, j ++) Entonces
      << matriz[i, j]
    Fin_Para
  Fin_Para
Fin
```

Seudocódigo

```
Algoritmo Trabajar con matrices
  Clase MatrizDemo
    Método principal()
      Declarar variables
        matriz : Arreglo[4][5] Entero
        i, j : Entero
      for(i = 0; i < 4; i ++)
        for(j = 0; j < 5; j ++)
          Solicitar valor
          Leer matriz[i][j]
        endfor
      endfor
      for(i = 0; i < 4; i ++)
        for(j = 0; j < 5; j ++)
          Imprimir matriz[i][j]
        endfor
      endfor
    Fin Método principal
  Fin MatrizDemo
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in)
var matriz = new int[4][5];

for(var i = 0; i < matriz.length; i++) {
   for(var j = 0; j < matriz[i].length; j++) {
      System.out.print("Valor: ");
      matriz[i][j] = l.nextInt();
   }
}
for(var i = 0; i < matriz.length; i++) {
   for(var j = 0; j < matriz[i].length; j++) {
      System.out.printf("%d\n", matriz[i][j]);
   }
}

for(var fila : matriz) {
   for(var value : fila) {
      System.out.printf("%d\n", value);
   }
}
```

Figura 27. Operaciones básicas con una matriz

Al igual que con los arreglos de una dimensión, en los de dos dimensiones, en el lenguaje de programación Java se pueden usar la propiedad *length* para conocer el número de filas de la matriz y por cada fila conocer la longitud de sus columnas, esto debido a que, en Java, una matriz es un arreglo de arreglos y cada fila podría tener un número diferente de columnas. También se puede usar un *forEach* para recorrer una matriz, el primer *for* obtiene cada fila (un arreglo), mientras que el segundo obtiene los valores (columnas).

Es momento de que intente resolver un problema construyendo un programa. El problema al que se enfrenta es calcular el total de medallas (oro + plata + bronce) que obtuvieron cada uno de los 16 cantones de la provincia de Loja, en las olimpiadas intercantonales. Para ello debe desarrollar un programa Java que devuelve el total. Es recomendable que intente usted solo resolver el problema propuesto

y luego compare su respuesta con el programa que se presenta más adelante.

Mini-especificación

```
Inicio
  medallero[16, 3], i[0 – n]
  i, i[0 – 15]
  j, i[0 – 3]
  //Total de medallas por provincia
  totalM, i[0 – n]
  Para(i <– 0, i < 16, i ++) Entonces
    Para(j <– 0, j < 3, j ++) Entonces
      << “Solicitar medalla”
      >> medallero[i, j]
    Fin_Para
  Fin_Para
  Para(i <– 0, i < 16, i ++) Entonces
    totalM <– 0
    Para(j <– 0, j < 3, j ++) Entonces
      totalM <– totalM + medallero[i, j]
    Fin_Para
    << totalM
  Fin_Para
Fin
```

Seudocódigo

```
Algoritmo Contar medallas
  Clase MedalleroProvincial
    Método principal()
      Declarar variables
        medallero: Arreglo[16][3] Entero
        i, j : Entero
        totalM : Entero
      for(i = 0; i < 16; i ++)
        for(j = 0; j < 3; j ++)
          Solicitar medalla
          Leer medallero[i][j]
        endfor
      endfor
      for(i = 0; i < 16; i ++)
        totalM = 0
        for(j = 0; j < 3; j ++)
          totalM = totalM + medallero[i][j]
        endfor
        Imprimir totalMedallas
      endfor
    Fin Método principal
  Fin MedalleroProvincial
Fin
```

Lenguaje de programación

```
Scanner l = new Scanner(System.in)
var medallero = new int[16][3];
int totalM;
for(var i = 0; i < medallero.length; i++) {
   for(var j = 0; j < medallero[i].length; j++) {
      System.out.print(“Medalla: ”);
      medallero[i][j] = l.nextInt();
   }
}
for(var provincia : medallero) {
   totalM = 0;
   for(var medalla : provincia) {
      totalM = totalM + medalla
   }
   System.out.printf(“%d\n”, totalM);
}
```

Figura 28. Programa para calcular el total de medallas

La Figura 28 muestra una posible solución al problema propuesto. Son 16 los cantones de la provincia de Loja, razón por la cual la matriz tendrá 16 filas y existen 3 tipos de medallas que se representan como columnas. Una vez ingresados, por cada fila se suman las 3 columnas (medallas) y se presenta esa suma. Es necesario que observe como en Java se puede usar el *forEach* para realizar los cálculos.

Luego de este par de semanas de estudio se concluye el tema de las estructuras de datos. En las siguientes dos semanas estudiará como crear sus propios métodos que luego podrá usar.

No olvide revisar y realizar las actividades de aprendizaje recomendadas, estas le ayudarán a entender de mejor manera cada uno de los temas que se han desarrollado en el transcurso de estas semanas.

Actividades de aprendizaje recomendadas

Responda a la siguiente Autoevaluación.

Autoevaluación 4

Para medir el nivel de comprensión de los temas que se desarrollaron en esta unidad, se propone la siguiente autoevaluación. Trate de resolver usted mismo los problemas, para luego revisar y comparar con el solucionario que se encuentran al final de este documento.

Responda a cada una de las preguntas planteadas a continuación que se basan en los contenidos propuestos en esta unidad. En esta evaluación se plantean preguntas de programación.

1. Un arreglo es una estructura de datos que:

    a. Únicamente puede contener valores de un mismo tipo de dato.
    b. Puede contener valores que pertenecen a varios tipos de dato.
    c. Puede contener valores de diferente tipo de dato, siempre y cuando esos tipos de dato sean *Entero y Real.*
    d. Puede contener valores de diferente tipo de dato, siempre y cuando esos tipos de dato sean *Cadena y Carácter*.

2. Un arreglo unidimensional ha sido declarado así: arr: Arreglo[10] Entero ¿cuáles serían los índices de la primera y última posición?

   a. Primera: 1, última 10
   b. Primera: 0, última 8
   c. Primera: 0, última 9
   d. Primera: 1, última 9

3. La variable que representa al índice o las posiciones de un arreglo generalmente es de tipo:

   a. *Carácter*
   b. *Cadena*
   c. *Real*
   d. *Entero*

4. Suponga que tiene un arreglo de 4 dimensiones, ¿cuántos ciclos repetitivos necesita para presentar los datos de ese arreglo?

   a. 1
   b. 2
   c. 3
   d. 4

5. ¿Cuál de las siguientes sentencias declara un arreglo de enteros de tamaño 10?

   a. *arr: Entero[9] Arreglo*
   b. *arr: Arreglo[10] Entero*
   c. *arr: Arreglo[9] Entero*
   d. *arr: Entero[10] Arreglo*

6. En las siguientes preguntas es necesario que escriba los programas en el seudocódigo estudiado hasta el momento.

7. Elabore un algoritmo que imprima los elementos de un arreglo de enteros (de tamaño: 7) que son pares.

8. Elabore un algoritmo que imprima los elementos de un arreglo (de tamaño: 10) que son números primos.

9. Elabore un algoritmo que calcule el promedio de los elementos de un arreglo (de tamaño: 9) e imprima aquellos valores que son mayores al promedio.

10. Elabore un algoritmo que encuentre el número mayor de todos los elementos que se encuentran en un arreglo (de tamaño 6) de números reales.

11. Se pidió a cuarenta estudiantes que califiquen la calidad de la información que brinda un centro universitario, en una escala del 0 al 9 (en donde 0 significa malo y 9 significa excelente). Coloque las 40 respuestas en un arreglo entero y elabore un algoritmo que sintetice los resultados de la encuesta a través un histograma con las frecuencias de cada respuesta.

Ir al solucionario

Revise el vídeo (videoconferencias, 2010) en el siguiente enlace: https://youtu.be/t3dcEC1bFD4.

Practique lo aprendido en esta unidad en el laboratorio de programación “Arreglos”, 2017) en el siguiente enlace: https://goo.gl/tdJS8S.

Revise el vídeo “Tutorial Java - 10. Matrices - Arreglos Bidimensionales” (Cofigofacilito, 2011) acerca del funcionamiento de los arreglos bidimensionales, en el siguiente enlace: https://youtu.be/SMIyVS3k848. Práctica lo aprendido en esta unidad mediante el laboratorio de programación “Matrices” “López, 2017) en el siguiente enlace: https://goo.gl/sGng1V.

Resultado de aprendizaje	Aplica las técnicas de descomposición estructurada (funcional) para dividir un programa en partes más pequeñas.

Contenidos, recursos y actividades de aprendizaje

Para las siguientes dos semanas aprenderá a reutilizar código escribiendo módulos. Estos módulos reciben diferentes nombres según el paradigma de programación que utilice, así se suelen denominar métodos en programación orientada a objetos. Este es el nombre que se usa en esta asignatura.

La base teórica para las siguientes dos semanas la encontrará en López Vargas, J. (2018) Guía Didáctica de Fundamentos de Programación. Loja – Ecuador: Editorial UTPL, en el apartado: “UNIDAD 5. MÉTODOS” y la parte práctica está en Lopez-Vargas, J. (2019) Java, apuntes básicos, en el apartado: “Métodos”.

A partir de aquí los contenidos son completamente nuevos para usted, ya que, en la asignatura de Algoritmos y resolución de problemas, no se cubren ni este ni el siguiente tema. Es por ello que el código que se presentará únicamente estará escrito en seudocódigo y Java.

Semana 11

Unidad 5. Métodos

Los métodos son una técnica que se viene usando desde hace mucho tiempo atrás, históricamente se remontan a lo que se denomina la programación modular, que consiste en dividir a un problema complejo en varios sub-problemas más simples de tal manera que sean lo suficientemente simples como para resolverlos fácilmente.

La técnica anterior se llama divide y vencerás (DYV) que ya era usada en el siglo XVIII. Actualmente también recibe la denominación de análisis descendente (Top-Down). Un método puede recibir muchos nombres, sin embargo, el concepto sigue siendo el mismo.

De manera general los métodos se pueden clasificar en dos grandes grupos: uno formado por aquellos métodos que regresan valor y otra

formado por aquellos que no regresan valor. Iniciará el estudio con el segundo tipo de métodos, pero antes es necesario que revise los estándares de programación que se recomienda para los métodos, para ello vaya a la página 22 del documento que está disponible en: https://goo.gl/EmPsj9 y busque la fila métodos. No olvide que un estándar de programación es únicamente un acuerdo consesuado que busca que todos los programadores lo adopten para mejorar la legibilidad de un programa.

5.1. Métodos que no regresan valor

En este primer grupo de métodos aprenderá conceptos como invocar a un método. Con ese fin es necesario que revise el siguiente recurso: "5.1. Métodos que no regresan valor" de López Vargas, (2018).

La invocación a un método puede verse como otra forma de modificar el flujo normal de ejecución de un programa, ya que cuando se encuentra la invocación a un método el flujo de ejecución se dirige al bloque de sentencias de ese método. El control se devuelve cuando termina la ejecución del método.

Recuerde también que un método puede estar compuesto por llamadas a otros métodos, es decir que pueden existir muchos "saltos" hasta que continúe el flujo de ejecución normal de un programa.

Es momento de estudiar el concepto métodos, desde el punto de vista de un lenguaje de programación, recuerde que el lenguaje seleccionado par la asignatura es Java. Es momento de que revise el recurso "Métodos" de Lopez-Vargas, (2019). La Figura 29 muestra como se puede crear e invocar métodos que no retornan valor en seudocódigo y Java.

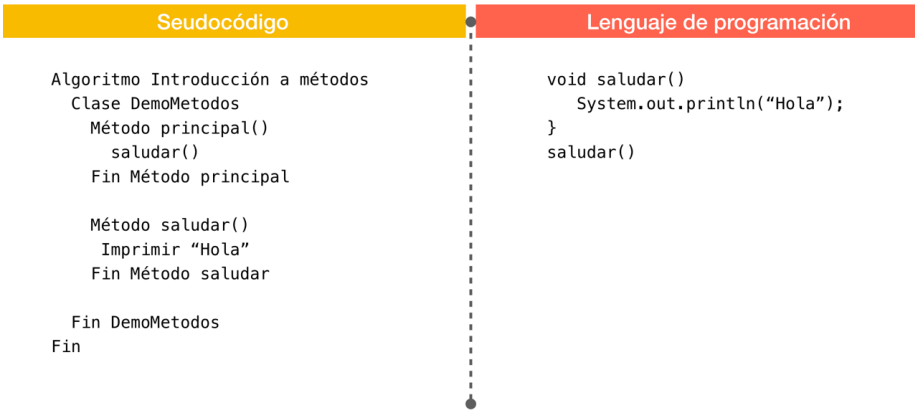


Figura 29. Ejemplo de un método no retorna valor

En el lenguaje de programación Java se utiliza la palabra reservada *void* para indicar que se trata de un método que no devuelve un valor. No así en seudocódigo, en donde no existe una palabra reservada o equivalente para marcar este tipo de métodos.

La invocación a los métodos consiste en ubicar el nombre de estos en la posición que se los necesite, convirtiéndose en una sentencia más. En seudocódigo un método se declara al terminar el método principal, en Java es igual, eso se ve claramente cuando se construyen clases, lamentablemente ese tema no es parte de la asignatura.

Dentro de los métodos existen otros conceptos que permiten que los métodos se conviertan en herramientas bastante útiles que por ejemplo pueden recibir datos para realizar su trabajo. La posición de estos elementos se muestra en la Figura 30. El detalle de cada uno de esos elementos se describe en el siguiente apartado.

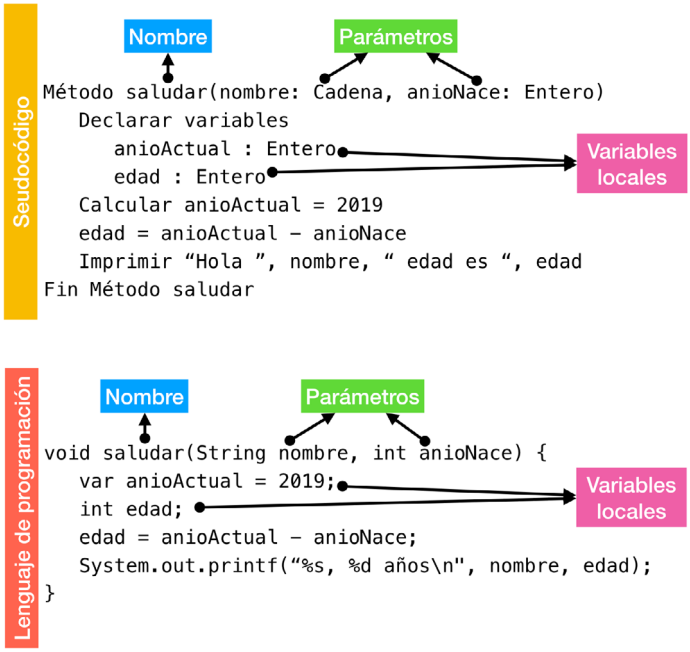


Figura 30. Parámetros y variables locales

5.2. Variables globales, locales y parámetros

Estos tres conceptos se asocian a los métodos en general, para conocer a detalle cada uno de estos temas es necesario que revise el recurso: "5.3. Variables locales, globales y parámetros". Luego de la lectura responda a la pregunta ¿Cuál es la diferencia entre variable local y parámetro?

Como pudo ver en la Figura 30 declarar un parámetro es similar a declarar una variable. La diferencia entre ellos es que en Java un parámetro no puede tener un valor inicial o valor por defecto, mientras que una variable si. Una variable local es una variable que se declara dentro de un método. Dicha figura no muestra una variable global, pero esta se resume en una variable que puede ser usada entre métodos.

La Figura 31 muestra la implementación completa del método saludar, esta vez recibiendo parámetros.

Seudocódigo

```
Algoritmo Introducción a métodos
  Clase DemoMetodos
    Método principal()
      Declarar variables
        nombre : Cadena
        anioNace : Entero
      Solicitar nombre y año nacimiento
      Leer nombre, anioNace
      saludar(nombre, anioNace)
    Fin Método principal

    Método saludar(nombre: Cadena, anioNace: Entero)
      Declarar variables
        anioActual : Entero
        edad : Entero
      Calcular anioActual = 2019
      edad = anioActual - anioNace
      Imprimir "Hola ", nombre, " edad es ", edad
    Fin Método saludar

  Fin DemoMetodos
Fin
```

Lenguaje de programación

```
void saludar(String nombre, int anioNace) {
   var anioActual = 2019;
   int edad;
   edad = anioActual - anioNace;
   System.out.printf("%s, %d años\n", nombre, edad);
}
Scanner l = new Scanner(System.in);
String nombre;
int anioNace;
System.out.print("Nombre: ");
nombre = l.next();
anioNace = l.nextInt();
saludar(nombre, anioNace);
```

Figura 31. Invocación y ubicación del método que recibe parámetros, pero no devuelve un valor.

Ponga atención en la invocación del método, *saludar*, observe como los parámetros son reemplazados por variables (valores). Aquí es importante considerar el tipo de dato de cada parámetro y el orden, es decir, hay que usar variables del mismo tipo de dato de los parámetros y escritos en el mismo orden en el que fueron declarados en el método.

Si bien los métodos que no devuelven valor son útiles para la didáctica, más útiles son los métodos que devuelven un valor. Ese es el tema que estudiará en el siguiente apartado.

5.3. Métodos que regresan valor

A diferencia de los que no regresan valor, este nuevo grupo de métodos no utilizan la palabra *void*, sino señalan uno de los tipos de

datos estudiados y además utilizan la sentencia *return* para señalar, dos cosas, la primera, el valor que el método devuelve y segundo, la finalización de la ejecución del método. ¿Cuántas sentencias *return* puede tener un método? Para responder a esta pregunta es necesario que revise el siguiente recurso.

"5.2. Métodos que regresan valor" de López Vargas, (2018). De seguro encontró la respuesta a la pregunta propuesta anteriormente. Ahora revise cómo el lenguaje de programación Java y el seudocódigo trabajan con este tipo de métodos, revisando el recurso: Lopez-Vargas, (2019) "Métodos que devuelven valor". La Figura 32 resume este tipo de métodos.

Seudocódigo

```
Algoritmo Introducción a métodos
  Clase Util
    Método principal()
      Declarar variables
        nro : Entero
      Solicitar número
      Leer nro
      if esPar(nro) == True then
         Imprimir "Es par"
      else
         Imprimir "No es par"
      endif
    Fin Método principal

    Método esPar(nro: Entero) : Booleano
      if nro MOD 2 == True then
         return True;
      else
         return False;
      endif
    Fin Método esPar
  Fin Util
Fin
```

Lenguaje de programación

```
boolean esPar(int nro) {
   return nro % 2 == 0;
}
Scanner l = new Scanner(System.in);
int valor;
valor = l.nextInt();
if(esPar(valor)) {
   System.out.printf("El %d es par\n", valor);
} else {
   System.out.printf("El %d NO es par\n", valor);
}
```

Figura 32. Ejemplo de un método que retorna un valor

Observe como a diferencia de los métodos anteriores, estos métodos señalan el tipo de dato que devolverán, en seudocódigo se ubica el tipo de dato, separado por dos puntos, luego del paréntesis de cierre de los parámetros, mientras que en Java se ubica el tipo de dato en lugar de *void*. Note también el uso de las sentencias *return*.

Cada sentencia *return* va acompañada de una variable o valor que es del mismo tipo de dato que el método señala que devolverá.

Aparentemente el método *esPar* se implementó de forma diferente en seudocódigo y Java, pero no es así son equivalentes y en Java es una simplificación ya que un operador relacional devuelve valores booleanos que es lo que señala que devolverá el método.

Antes de finalizar esta semana es necesario que trate de resolver un problema construyendo un programa en el lenguaje de programación Java. El problema es el siguiente, se necesita construir un método que determine si un número es primo o no. Trate de implementar usted mismo el programa. Más adelante encontrará una solución a ese problema. La Figura 33 muestra una solución al problema propuesto.

Seudocódigo

```
Algoritmo Método para determinar Primo
  Clase Util
    Método principal()
      Declarar variables
        nro : Entero
      Solicitar número
      Leer nro
      if esPrimo(nro) == True then
         Imprimir "Es primo"
      else
         Imprimir "No es primo"
      endif
    Fin Método principal
    Método esPrimo(nro: Entero) : Booleano
      Declarar variables
        div : Entero
        esPrimo : Booleano
      esPrimo = True
      if nro > 1 then
        for(div = 2; div < nro; div ++)
          if nro MOD div == 0 then
            esPrimo = False
          endif
        endfor
        return esPrimo
      else
        return False
      endif
    Fin Método esPrimo
  Fin Util
Fin
```

Lenguaje de programación

```
boolean esPrimo(int nro) {
   var esPrimo = true;
   if(nro > 1) {
      for(var div = 2; div < nro; div ++) {
         if(nro % div == 0) {
            esPrimo = false;
         }
      }
      return esPrimo;
   } else {
      return false;
   }
}
Scanner l = new Scanner(System.in);
int valor;
valor = l.nextInt();
if(esPrimo(valor)) {
   System.out.printf("El %d es primo\n", valor);
} else {
   System.out.printf("El %d NO es primo\n", valor);
}
```

Figura 33. Método que determinar si un número es par

Si tiene algún problema para implementar el programa propuesto o con algún contenido por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o el espacio de tutoría permanente.

La próxima semana revisará el concepto de recursividad que es una estrategia que bastante utilizada cuando se trabaja con métodos y conocer algunos criterios para la evaluar métodos y determinar si son o no métodos reutilizables.

Semana 12

5.4. Recursividad

Como revisó anteriormente un método puede llamar a otro, esto abre la posibilidad que un método puede llamarse a sí mismo, a esto es lo que se denomina una llamada recursiva. A primera vista esto no tiene mucho sentido, pero esta técnica es bastante interesante y usada. Los detalles teóricos de este los encontrará en “5.4 Recursividad” de López Vargas, (2018).

Recuerde que para construir un método recursivo es necesario contar con dos casos, uno de ellos se denomina el caso base y otro el caso recursivo, este último permite una nueva iteración, mientras que el primero termina con las iteraciones.

Para comprender como se implementa la recursividad en un lenguaje de programación debe revisar el recurso Lopez-Vargas, (2019) “Recursividad”.

En la Figura 34 encontrará un programa escrito en seudocódigo y Java que implementa recursividad.

Seudocódigo

```
Algoritmo Método recursivo cálculo factorial
  Clase Util
    Método principal()
      Declarar variables
        nro : Entero
        factorial : Entero
      Solicitar número
      Leer nro
      factorial = calcularFactorial(nro)
      Imprimir factorial
    Fin Método principal
    Método calcularFactorial(nro: Entero) : Entero
      if nro <= 1 then
        return 1
      else
        return nro * calcularFactorial(nro - 1)
      endif
    Fin Método calcularFactorial
  Fin Util
Fin
```

Lenguaje de programación

```
int calcularFactorial(int nro) {
   if(nro <= 1) {
      return 1;
   } else {
      return n * calcularFactorial(n - 1);
   }
}
Scanner l = new Scanner(System.in);
int valor;
valor = l.nextInt();
var factorial = calcularFactorial(valor);
System.out.printf("%d\n", factorial);
```

Figura 34. Cálculo del factorial de un número usando recursividad

La recursividad se utiliza principalmente en el recorrido de estructuras de datos como árboles ya que permite que el código sea más legible.

Es momento que intente resolver un problema utilizando recursividad. El problema el contar cuántos dígitos tiene un número, por ejemplo: 345 tiene 3 dígitos, 15234 tiene 5 dígitos. Recuerde que su programa debe estar escrito en el lenguaje de programación Java.

Una posible solución al problema propuesto se muestra en la Figura 35. En la implementación hecha en Java se llama al método en la sentencia que se utiliza para imprimir.

Seudocódigo

```
Algoritmo Método recursivo para contar dígitos
  Clase Util
    Método principal()
      Declarar variables
        nro : Entero
        nroDigitos : Entero
      Solicitar número
      Leer nro
      nroDigitos = contarDigitos(nro)
      Imprimir nroDigitos
    Fin Método principal
    Método contarDigitos(nro: Entero) : Entero
      if nro / 10 == 0 then
        return 1
      else
        return 1 + contarDigitos(nro / 10)
      endif
    Fin Método contarDigitos
  Fin Util
Fin
```

Lenguaje de programación

```
int contarDigitos(int nro) {
   if(nro /10 == 0) {
      return 1;
   } else {
      return 1 + contarDigitos(nro / 10);
   }
}
Scanner l = new Scanner(System.in);
int valor;
valor = l.nextInt();
System.out.printf("%d\n", contarDigitos(valor));
```

Figura 35. Método recursivo para contar el número de dígitos que tiene un número

Ya conoce los diferentes tipos de métodos y también estudió los métodos recursivos, posiblemente se esté preguntado ¿cómo puedo evaluar si mi método es reutilizable? La respuesta a esta pregunta la encontrará en el siguiente apartado.

5.5. Criterios para evaluar métodos

Para conocer los criterios que puede usar para evaluar si un método es reutilizable los puede encontrar en "5.5. Criterios para evaluar métodos" de López Vargas, (2018).

Una vez hecha la lectura anterior responda a la pregunta ¿Cómo diferenciar entre parámetros y variables locales? Seguro que encontró la respuesta a esta pregunta, recuerde que los parámetros son la información que necesita el método para trabajar, esta información generalmente se envía desde fuera. Las variables locales, únicamente tienen utilidad dentro del método y no fuera de el.

Si tiene algún problema con algún contenido por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o el espacio de tutoría permanente.

La próxima semana revisará el tipo de dato *String* y aprenderá a trabajar con ciertos métodos especializados para este tipo de datos.

No olvide revisar y realizar las actividades de aprendizaje recomendadas, estas le ayudarán a entender de mejor manera cada uno de los temas que se han desarrollado en el transcurso de estas semanas.

Actividades de aprendizaje recomendadas

Responda a la siguiente Autoevaluación.

Autoevaluación 5

Para comprobar que ha asimilado correctamente el tema desarrolle la siguiente autoevaluación. Recuerde que al final de este documento encontrará el solucionario a las preguntas propuestas.

Responda a cada una de las preguntas planteadas a continuación que se basan en los contenidos propuestos en esta unidad. En esta evaluación se plantean preguntas de programación.

1. ¿Cuál de las siguientes palabras se tiene que utilizar cuando se define un método que retorna un valor?

   a. Calcular
   b. Entero
   c. *return*
   d. Arreglo

2. ¿Cuál de las siguientes porciones de código corresponden a un método que devuelve un valor?

   a. *Método a(x: Entero): Entero*
   b. *Entero: Método a(x: Entero)*
   c. *Método a(x: Entero)*
   d. *Método a: Entero (x: Entero)*

3. Una variable es local cuando se ubica en las declaraciones de:

   a. Algoritmo
   b. Clase
   c. Método

4. Un parámetro se puede usar únicamente dentro de:

   a. Un algoritmo
   b. Una clase
   c. Un método

5. ¿Cuántas sentencias return debe tener un método?

   a. Sólo una
   b. Sólo dos
   c. Máximo tres
   d. Las que sean necesarias

Para las siguientes preguntas es necesario que escriba programas utilizando el seudocódigo estudiado hasta el momento.

6. Elabore un algoritmo que a través de un menú permita a un usuario calcular el factorial y elevar un número a una determinada potencia. Utilice métodos para ambos casos.

7. Escriba un algoritmo que posea 2 métodos, el primero de ellos que permita determinar si un número es par. El segundo método que calcule el número impar según su posición. Ejemplo: el usuario ingresa la posición 1 el método debe devolver 1, posición 2 se devuelve 3, posición 3 se devuelve 5 y así sucesivamente).

8. Elabore un algoritmo que a través del uso de métodos devuelva un número mayor a un número entero positivo que es enviado como parámetro, pero con la condición de que el número devuelto sea primo. Otro método que según una posición devuelve un término de la serie de Fibonacci que corresponda (Ejemplo: posición 1, Fibonacci 0; posición 2, Fibonacci 1; posición 3, Fibonacci 1; posición 4, Fibonacci 2; etc.). Únicamente construya los métodos, deje de lado los otros elementos.

9. Elabore un algoritmo que permita elevar un número a una potencia utilizando un método recursivo. Utilice el siguiente ejemplo

$$5^4 = 5^* \ 5^* \ 5^* \ 5 = 5^* \ 5^3$$
$$5^3 = 5^* \ 5^* \ 5 = 5^* \ 5^2$$
$$5^2 = 5^* \ 5 = 5^* \ 5^1$$
$$5^1 = 5 = 5^* \ 5^0$$
$$5^0 = 1 = 5^* \ 1$$

10. Elabore un algoritmo que utilizando recursividad encuentre el divisor de una división de enteros. Para ello utilice restas sucesivas así: se quiere dividir 5 para 2

Tabla 3. Explicación de la división por recursividad

Ejemplo	Regla	Ejemplo
5 – 2 = 3 (1)	Se resta hasta que el resultado es negativo.	dividir (x, y) Si (x – y) < 0 devolver 0 Sino devolver 1 + dividir (x – y, y) 3 – 2 = 1 (1) 1 – 2 = -1 (0)

Ir al solucionario

Revise el vídeo “Métodos” (Videoconferencias, 2010) , contiene información acerca de los métodos. Videoconferencias. (2010) Métodos. [Archivo de video]. Recuperado de https://youtu.be/QRrAa8k-Las.

Practique lo aprendido en esta unidad mediante el laboratorio “Métodos” (López, 2017), en el siguiente enlace: https://goo.gl/gFKFQB.

Revise el vídeo “Fundamentos de la programación I – Recursividad – Fernando Pereñiguez” (UCAM, 2014), en el siguiente enlace: https://goo.gl/3hp37c. Este contiene información acerca de los métodos recursivos. Practique lo aprendido en esta unidad mediante el laboratorio de programación “Recursividad” (López, 2017), en el siguiente enlace https://goo.gl/4et8GG.

Resultado de aprendizaje	Escribe programas que utilizan cada una de las siguientes estructuras de datos: cadenas.

Contenidos, recursos y actividades de aprendizaje

Para las siguientes dos semanas estudiará sobre el manejo del tipo de datos *String* y conceptos básicos sobre archivos. La variables que son cadenas de texto cuentan con una amplia variedad de métodos que permiten realizar muchas funciones sobre este tipo de dato.

En cuanto al manejo de archivos, el seudocódigo ayudará con los conceptos básicos y se deja de lado el desarrollo de programas que manipulen archivos, esa actividad queda de uso exclusivo para el lenguaje de programación.

La base teórica para semana la puede encontrar en: López Vargas, J. (2018) Guía Didáctica de Fundamentos de Programación. Loja

– Ecuador: Editorial UTPL, en el apartado: “UNIDAD 6. FUNCIONES ESTÁNDAR Y ARCHIVOS”. Mientras que la base práctica se encuentra en Lopez-Vargas, J. (2019) Java, apuntes básicos, en los apartados: “Cadenas de texto” y “Archivos”.

Semana 13

Unidad 6. Funciones estándar y archivos

Para finalizar los contenidos de la asignatura se propone el estudio de los temas funciones estándar y archivos. Las funciones estándar son métodos que generalmente se encuentran en los lenguajes de programación, mientras que los archivos son una forma para almacenar información. Estos conceptos lo preparan de mejor manera en sus conocimientos de los fundamentos de la programación.

En esta semana se concentrará en el estudio de las funciones estándar, especialmente en aquellos que se utilizan o aplican en el tipo de dato cadena de caracteres.

6.1. Funciones estándar

Es el momento de revisar las funciones estándar, para ello es necesario que realice una lectura comprensiva del tema "6.1. Funciones estándar" de López Vargas, (2018).

Así como existen funciones que trabajan con datos numéricos, existen una gran variedad de métodos que trabajan con otros tipos de datos. Es de nuestro interés que se revisen los métodos que tienen que ver con el manejo de variables tipo *String*, en el siguiente apartado conocerá el detalle de estos métodos.

6.2. Funciones sobre cadena de caracteres

¿Cuál es el concepto clave para entender a estas funciones? Para dar respuesta a esta pregunta es necesario que revise el siguiente recurso: "6.2. Funciones sobre cadena de caracteres" de López Vargas, (2018).

De seguro encontró la respuesta a esta pregunta. Una variable de tipo *String*, se debe considerar como un arreglo de caracteres. Con este enfoque se pueden entender varias de las funciones que revisó anteriormente.

Muchas de las funciones que se encuentran a nivel de seudocódigo tienen su equivalencia en los lenguajes de programación. Para comprender este tema es necesario que revise el siguiente recurso "Cadenas de texto" en Lopez-Vargas, (2019).

Una vez que ha revisado la parte teórica y práctica se resumen en la siguiente tabla muestra esa equivalencia entre funciones y métodos.

Tabla 4. Equivalencia de funciones de seudocódigo y métodos Java

Seudocódigo	Java
Carácter	charAt
Ordinal	(int)'a'
Concatenar	concat +
Subcadena	substring
Borrar	replace
Longitud	Length
Posición	indexOf
Cadena	String.valueOf

La mayoría de las funciones de seudocódigo tienen un método equivalente, mientras que otras se resuelven con sentencias como ordinal o concatenar usando el operador "+".

No olvide que el lenguaje de programación Java considera al tipo de dato *String* como una clase y cada variable de ese tipo es un objeto de esa clase. Esta es la razón por la cual no se denominan funciones sino métodos y en las llamadas a estos se utiliza la notación <variable>.<método> así como muestra la Figura 36.

Lenguaje de programación

```
String nombre = "Jorge";
var nroLetras = nombre.length();
System.out.printf("%s tien %d letras", nombre, nroLetras);
```

Figura 36. Invocación a un método de la clase String

Dentro de los lenguajes de programación existe lo que se denomina un API que se traduce como un Interfaz de Programación de Aplicaciones o Application Programming Interface, que es básicamente un conjunto de funciones o métodos previamente construidos y que se integran al lenguaje de programación.

El lenguaje de programación Java cuenta con su propio API y la documentacióbn del mismo la puede encontrar aquí: https://docs.oracle.com/en/java/javase/ Es conveniente que seleccione el link bajo el título Current Release. Luego podrá navegar por los diferentes módulos que forman el lenguaje de programación. Para una visión básica recomiendo que seleccione el módulo *java.base*.

Una vez que conoce sobre el API y la documentación del mismo responda a la pregunta ¿Qué otros métodos posee el tipo de dato String? De seguro pudo encontrar esa clase en la documentación, sino fue así use el siguiente link: https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/String.html. No olvide que los lenguajes de programación están en constante evolución, es por ello que se recomienda revisar siempre la última versión del API.

Es momento de resolver un problema. Suponga que ha desarrollado un programa en el cual el usuario debe ingresar 10 números reales y presenta la suma de todos los números ingresados. Para facilidad del usuario se puede utilizar como separador decimal, la coma o el punto. Internamente su programa debe transformar la coma en punto, ya que el programa ha sido desarrollado en Java.

Le sugiero que lea cada valor ingresado como una cadena de texto, busque si existe la coma, la reemplace, transforme a real y haga las operaciones. Intente resolver usted mismo el problema y luego compare su solución con la que se presenta a continuación.

Lenguaje de programación

```
Scanner l = new Scanner(System.in);
var suma = 0.0;
String nroTxt;
var cont = 0;
var nro = 0.0;
while(cont < 10) {
   System.out.print("Ingrese un número: ");
   nroTxt = l.next();
   if(nroTxt .indexOf(',') != -1) {
      nroTxt = nroTxt.replace(",", ".");
   }
   nro = Double.valueOf(nroTxt);
   suma = suma + nro;
   cont = cont + 1;
}
System.out.printf("%f\n", suma);
```

Figura 37. Implementación de una solución al problema propuesto

La figura anterior muestra una posible solución, aquí se utiliza algunos métodos de la clase *String* y un de la clase *Double* para convertir un texto a número.

Si tuvo algún problema para comprender este programa o para construirlo, por favor comuníquese con su tutor, él le ayudará a resolverlos. Los medios de comunicación que dispone son mensajería interna del entorno, correo electrónico o en el espacio de tutoría permanente.

De esta forma se cierra esta semana, en donde conoció algo del API del lenguaje de programación Java. La siguiente semana estudiará teóricamente a los archivos y pondrá en práctica esos conceptos manipulando de forma básica un archivo de texto.

Semana 14

6.3. Archivos

¿Cuáles son los conocimientos teóricos básicos que se necesitan para manejar archivos dentro de un programa? Para dar respuesta a esta pregunta, es necesario que revise el siguiente recurso: "6.3. Archivos " de López Vargas, (2018), por favor realice una lectura comprensiva de esos conceptos.

Ahora ya conoce que detrás de un archivo existen dos conceptos como son los campos y los registros. La Figura 38 muestra gráficamente como se encuentra estructurado un archivo.[FCE: Insertar la imagen: archivos1]

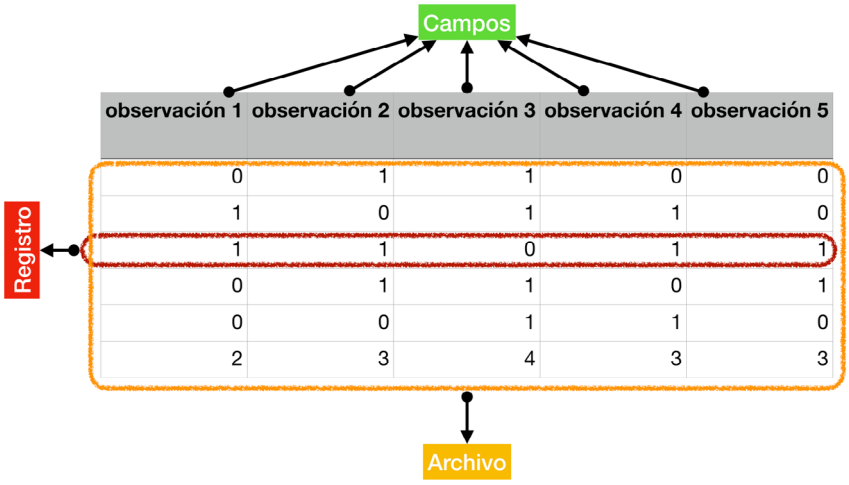


Figura 38. Resumen de la estructura básica de un archivo.

Las cabeceras que identifican cada uno de los campos generalmente no son parte de los registros, en la figura anterior se los utilizó únicamente con carácter explicativo.

Como ya lo revisó existen muchas funciones para el manejo de archivos, son tantas que pueden resultar abrumadoras es por ello que se han seleccionado como importantes las siguientes: crear, abrir y leer un archivo. Así como muestra la tabla 5 de López Vargas, (2018).

Es momento de trabajar con un lenguaje de programación para que muestre de forma práctica como manipular un archivo. Con este objetivo en mente revise el recurso “Archivos” en Lopez-Vargas, (2019).

Para resumir, lo aprendido con Java se propone el siguiente ejercicio. Elabore un programa en Java que lea y presente en pantalla la información que se encuentra en un archivo separado por comas y que agregue información al mismo.

Para resolver este problema es necesario que exista el archivo y que tenga cierta información cargada previamente, por lo que se sugiere que cree un archivo con el nombre *datos.csv* y lo grave en algún directorio de su computador. No olvide al ruta de ese archivo. El contenido del archivo puede ser:

0, 1, 1, 0, 0
1, 0, 1, 1, 0
1, 1, 0, 1, 1
0, 1, 1, 0, 1
0, 0, 1, 1, 0
2, 3, 4, 3, 3

Ahora se muestra una solución al problema propuesto. En el programa Java que se muestra en la Figura 39 se escribieron dos métodos, uno que permite leer el contenido del archivo línea a línea y presentarlo en pantalla. El segundo método agrega la información a ese archivo.

Lenguaje de programación

```
void leerArchivo(String ruta) {
    try {
       var lineas = Files.lines(Paths.get(ruta));
       lineas.forEach(System.out::println);
    } catch (IOException e) {
       e.printStackTrace();
    }
}
void agregarDatos(String ruta) {
   String newContent = "\n2, 3, 4, 3, 3";
   try {
      Files.write(Paths.get(ruta), newContent.getBytes(), StandardOpenOption.APPEND);
   } catch(IOException e) {
      e.printStackTrace();
   }
}
var rutaAArchivo = “su_ruta_al_archivo_datos.csv”;
leerArchivo(rutaAArchivo);
agregarDatos(rutaAArchivo);
```

Figura 39. Programa Java que resuelve el problema propuesto

Es posible que ahora mismo se esté preguntando que son esas sentencias *try* y *catch*. Estas sentencias permiten manejar los errores que se pueden presentar al trabajar con archivos como por ejemplo que el archivo no exista. En el bloque *try* se ubica la sentencia que posiblemente lance una excepción en el *catch* las sentencias que se ejecutarán si se lanzó una excepción.

Ambos métodos utilizan la ruta del archivo, no se olvide de configurar la variable *rutaAArchivo* con su ruta al archivo. Para acceder a ese archivo utiliza la clase *Paths* y el método *get*, que son parte del API de Java.

Para la lectura, se utiliza el método *lines* de la clase *Files* que básicamente permite obtener cada una de las líneas que se encuentran dentro del archivo. Luego utiliza el método *forEach* para imprimirlas, esta acción se hace utilizando programación funcional, recuerde que Java tiene soporte para ese paradigma de programación.

Para la escritura se utiliza la clase *Files*, pero esta vez el método *write*. Ese método necesita la ruta del archivo, el contenido (como bytes) y la operación que se realizará que es agregar el contenido al archivo, es por ello que se ubica la operación *APPEND*.

Ahora conoce los conceptos básicos del manejo de archivos tanto teóricamente como de forma práctica. Recuerde que este tema es únicamente introductorio es por ello que no se revisarán otros conceptos.

No olvide revisar y realizar las actividades de aprendizaje recomendadas, estas le ayudarán a entender de mejor manera cada uno de los temas que se han desarrollado en el transcurso de estas semanas.

Actividades de aprendizaje recomendadas

Responda a la siguiente Autoevaluación.

Autoevaluación 6

Para comprobar que ha asimilado correctamente el tema desarrolle la siguiente autoevaluación. Recuerde que al final de este documento encontrará el solucionario a las preguntas propuestas.

Responda a cada una de las preguntas planteadas a continuación que se basan en los contenidos propuestos en esta unidad. En esta evaluación se plantean preguntas de programación.

1. Dentro de las funciones de cadena de caracteres, es fundamental comprender que una variable tipo Cadena, puede trabajarse como si fuera un:

   a. *Entero*
   b. *Real*
   c. *Booleano*
   d. *Arreglo*

2. La función *Longitud* permite obtener:

   a. Un carácter representado por un valor entero.
   b. El número de caracteres que tiene la cadena.
   c. Una copia reducida de la cadena.
   d. El número que está representado por la cadena.

3. Existe una variable de tipo cadena que se llama nombre y contiene el valor “María del Carmen”. La llamada a la siguiente función: SubCadena(nombre, 5, 3) ¿qué valor devuelve?

   a. María
   b. ría
   c. del
   d. Carmen

4. La sentencia Longitud(“El arte vive en Loja”) ¿qué valor devuelve?

   a. 22
   b. 20
   c. 19
   d. 16

5. Un registro es un

   a. Conjunto de archivos
   b. Archivo
   c. Conjunto de datos
   d. Permiso de escritura

6. Señale las características de un archivo secuencial

   a. Es muy eficaz para el proceso en lotes.
   b. Se almacenan en una posición relativa fija.
   c. Se utiliza una llave para tener acceso al archivo.
   d. La búsqueda es poco eficiente.

7. El “modo” es un mecanismo que señala

   a. Los permisos de apertura de un archivo.
   b. La cantidad de bytes que tendrá un registro.
   c. La ruta física del archivo
   d. El tipo de organización de un archivo.

8. Un archivo es una colección de:

   a. Datos
   b. Variables
   c. Registros
   d. Directorios

En las siguientes preguntas es necesario escribir programas utilizando el seudocódigo estudiado hasta el momento.

9. Elabore un algoritmo que permita leer un número entero e indique si es capicúa (un número es capicúa, si este se lee igual de izquierda a derecha que en sentido contrario). Utilizar un método que reciba como parámetro el número y que devuelva TRUE si es capicúa o FALSE en caso contrario. Ejemplo: 1991.

10. Elabore un algoritmo que trabaje con un arreglo palabras, que almacene hasta 7 palabras. El algoritmo debe tener un método que devuelve la posición de la palabra más extensa (mayor número de caracteres). El algoritmo debe imprimir únicamente la palabra más extensa.

Ir al solucionario

Ponga en práctica lo aprendido en esta unidad mediante los siguientes laboratorios de programación:

- "Funciones para cadenas de texto" (López, 2017), en el siguiente enlace: https://goo.gl/SW7KEv.
- "Matrices" (López, 2017),, en el siguiente enlace: https://goo.gl/GSb54t.

Actividades finales del bimestre

Semana 15

En esta semana es recomendable que vuelva a revisar los contenidos que se desarrollaron en las semanas 9, 10 y 11.

No olvide de practicar creando varios programas utilizando especialmente el lenguaje de programación Java.

Si tiene algún problema, por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o en el espacio de tutoría permanente.

Semana 16

Para esta última semana de preparación se recomienda que haga una revisión de los contenidos que se cubrieron en las semanas 12, 13 y 14.

No olvide de practicar creando varios programas utilizando especialmente el lenguaje de programación Java.

Si tiene algún problema, por favor comuníquese con su tutor, ya sea por mensajería interna de este entorno, correo electrónico o en el espacio de tutoría permanente.

4. Solucionario

Primer bimestre

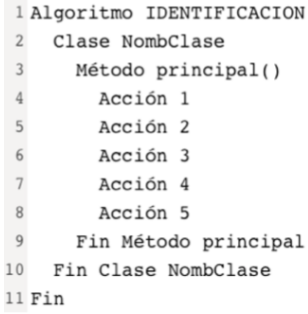
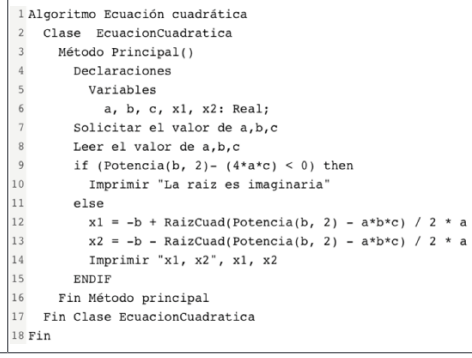
Autoevaluación 1		
Pregunta	**Respuesta**	**Retroalimentación**

1	`Declaraciones`
	`  Variables`
	`  nombre:cadena`
	`  edad:Entero`
	`  estatura:Real`
	`  mayorEdad:Booleano`
	`  genero:Caracter`
		Una solución podría ser la que se muestra a continuación. Recuerde que para declarar una variable es necesario contar con el nombre de esta y el tipo de dato. La declaración se la hace en la sección de variables.

2	A	Se recomienda seguir el estándar de programación del lenguaje Java, también conocido como "*camel case*".
3	C	Según el estándar utilizado, una constante se escribe en mayúsculas y si está formada por dos palabras, estas se separan a través de un guion bajo.
4	D	El requisito para que un valor sea almacenado en una variable de tipo Carácter es que contenga un único elemento. En este caso contiene un solo dígito.
5	C	Un número entero no posee parte decimal. No olvide que el separador decimal es el punto.
6	B	Es el único valor numérico de la lista.
7	A	Por teoría de conjuntos, todos los enteros son reales.

Autoevaluación 1		
Pregunta	**Respuesta**	**Retroalimentación**
8	E	Los únicos valores de una variable booleana son verdaderos o falsos.
9	A	En este caso se necesita utilizar paréntesis para agrupar las operaciones, siendo este literal la única agrupación válida.
10	D	El residuo de esa división entera es igual a 2.

Ir a la autoevaluación

Autoevaluación 2		
Pregunta	**Respuesta**	**Retroalimentación**
1	C	El orden normal en el que se ejecuta un programa es una sentencia tras otra de forma secuencial.
2	A	La única forma para modificar el flujo de ejecución normal de un programa es a través de las estructuras de control.
3	A	El método principal siempre va dentro de una clase y esta va dentro del algoritmo.
4	A	A través de exponentes fraccionarios se puede calcular raíces.
5		La estructura base se muestra a la izquierda. Recuerde que un algoritmo debe tener una descripción, una clase y un método principal, en donde se escriben las acciones.
6		Una posible solución es la que se muestra en la parte izquierda, ya que se pide el ingreso de los datos que señala el problema y se presentan ambas soluciones.

Autoevaluación 2

Pregunta	Respuesta	Retroalimentación

7	`Algoritmo Calcular el baricentro`
	`  Clase Baricentro`
	`  Método Principal`
	`  Declaraciones`
	`  Variables`
	`  ax, ay, bx, by, cx, cy: Enteros`
	`  gx, gy: reales`
	`  Solicitar A(x,y)`
	`  Leer ax, ay`
	`  Solicitar B(x,y)`
	`  Leer bx, by`
	`  Solicitar C(x,y)`
	`  Leer cx, cy`
	`  Calcular gx=(ax+bx+cx)/3`
	`  Calcular gy=(ay+by+cy)/3`
	`  Imprimir "El valor de G(x,y)", gx, gy`
	`  Fin Método principal`
	`  Fin Clase Baricentro`
	`Fin`
		El programa que resuelve el problema debe pedir el ingreso de las coordenadas x, y de los 3 puntos, aplicar la formular y realizar las operaciones que muestra la fórmula. El siguiente programa cumple con esos requisitos.

8	`Algoritmo Obtener día de la semana`
	`  Clase  DiaSemana`
	`  Método Principal ()`
	`  Declaraciones`
	`  Variables`
	`  dia : Entero`
	`  nomDia: Cadena`
	`  Solicitar el día`
	`  Leer dia`
	`  switch (letra) es:`
	`  1: Asignar nomDia="Lunes"`
	`  2: Asignar nomDia ="Martes"`
	`  3: Asignar nomDia ="Miercoles"`
	`  4: Asignar nomDia ="Jueves"`
	`  5: Asignar nomDia ="Viernes"`
	`  6: Asignar nomDia ="Sabado"`
	`  7: Asignar nomDia ="Domingo"`
	`  default: "Número fuera de rango"`
	`  endswitch`
	`  Imprimir "El nombre del dia",  NomDia`
	`  Fin del método principal`
	`  Fin de la clase`
	`Fin`
		La forma más sencilla de resolver este problema es utilizando un SWITCH, que evalué el número de entrada y que escoja el nombre del día. Observe que también se controla el caso en el que se ingresa un valor fuera del rango permitido.

Autoevaluación 2

Pregunta	Respuesta	Retroalimentación

9	```
	Algoritmo Calcular el índice de masa corporal
	  Clase IndiceMasaCorporal
	  Método principal ()
	  Declaraciones
	  Variables
	  peso, altura, imc : Real
	  sexo : Carácter
	  Solicitar peso, estatura, sexo
	  Leer peso, altura, sexo
	  Calcular imc = peso / (altura * altura)
	  Imprimir "Indice masa corporal=", imc
	  if (sexo == 'H') then
	  if imc <= 15 THEN
	  Imprimir "Delgadez pronunciada"
	  else
	  if (imc > 15 AND imc < 20) then
	  Imprimir "Delgadez"
	  else
	  if (imc > 20 AND imc < 25) then
	  Imprimir "Normalidad"
	  else
	  if (imc > 25 AND imc < 30) then
	  Imprimir "Gordura"
	  else
	  if (imc >= 30) then
	  Imprimir "Obesidad"
	  endif
	  endif
	  endif
	  endif
	  endif
	  else
	  if (imc <= 13) then
	  Imprimir "Delgadez pronunciada"
	  else
	  if (imc > 13 AND imc < 18) then
	  Imprimir "Delgadez"
	  else
	  if (imc > 18 AND imc < 23) then
	  Imprimir "Normalidad"
	  else
	  if (imc > 23 AND imc < 28) then
	  Imprimir "Gordura"
	  else
	  if (imc >= 28) then
	  Imprimir "Obesidad"
	  endif
	  endif
	  endif
	  endif
	  endif
	  endif
	  Fin Método principal
	  Fin Clase IndiceMasaCorporal
	Fin
	``
		La solución a este problema necesita de 3 datos de entrada, género, altura y edad; el siguiente paso es calcular el IMC aplicando la fórmula. Luego clasificar ese valor según el género y el valor del IMC, para lo que se utiliza varias sentencias de selección.

Autoevaluación 2

Pregunta	Respuesta	Retroalimentación

10	Algoritmo CALCULAR EDAD
	  Clase Edad
	  Método principal()
	  Declaraciones
	  Variables
	  añoAc, mesAc, diaAc : Entero
	  añoNa, mesNa, diaNa: Entero
	  añosE, mesesE, diasE: Entero
	  Solicitar año, mes y día actual
	  Leer añoAc, mesAc, diaAc
	  Solicitar año, mes y día de nacimiento
	  Leer añoNa, mesNa, diaNace
	  if (añoNa <= añoAc) then
	  if (diaAc < diaNa) then
	  Calcular diaAc = diaAc + 30
	  Calcular mesAc = mesAc -1
	  endif
	  if (mesAc < mesNa) then
	  Calcular mesAc = mesAc + 12
	  Calcular añoAc = añoAc – 1
	  endif
	  Calcular añosE = añosAc – añoNa;
	  Calcular mesesE = mesAc – mesNa
	  Calcular diasE = diasAc – diaNa
	  Imprimir añosE, mesesE, diasE
	  else
	  Imprimir “Datos incorrectos”
	  endif
	  Fin método principal
	  Fin Clase Edad
	Fin
		Las entradas son la fecha actual y la fecha de nacimiento de una persona, el ingreso de estos valores se hace de forma individual. El algoritmo es básico y primero se debe validar que el año de nacimiento sea menor que el actual. Luego se hacen operaciones de resta con los días y los meses.

Ir a la autoevaluación

Autoevaluación 3		
Pregunta	**Respuesta**	**Retroalimentación**
1	B	Este es un ciclo que se ejecuta por lo menos una vez, por lo que la condición se ubica al final.
2	C	Según el tipo de problema la condición podría ser simple o compuesta.
3	B	Controlar un ciclo repetitivo está en relación directa con la condición de que debe cumplirse para que se repitan las sentencias, por lo que la variable que controla el ciclo debe estar en la condición.
4	B	Este ciclo tiene la condición al final, por lo que primero se ejecutan las sentencias y luego se evalúa la condición, lo que garantiza la ejecución de por lo menos una vez.
5	A	En un ciclo *for* el incremento o decremento de la variable que controla el ciclo es parte de su declaración, es así que modificar su valor en otro sitio se considera una mala práctica.

Autoevaluación 3

Pregunta	Respuesta	Retroalimentación

6	`Algoritmo Suma menores`
	`  Clase Sumatoria`
	`  Método principal()`
	`  Declaraciones`
	`  Variables`
	`  limite, contador, suma, nro : Entero`
	`  Calcular contador = 0`
	`  Calcular suma = 0`
	`  Calcular nro = 1`
	`  Solicitar el límite`
	`  Leer limite`
	`  do`
	`  suma = suma + nro`
	`  nro = nro + 1;`
	`  contador = contador + 1`
	`  while (contador < limite)`
	`  Imprimir suma`
	`  Fin Método principal`
	`  Fin Clase Sumatoria`
	`Fin`
		El programa debe recibir como entrada el límite superior, luego debe generar todos los números comprendidos entre 1 y ese límite. Además, debe usar un acumulador para almacenar la suma. La salida del programa es la suma de esos números.

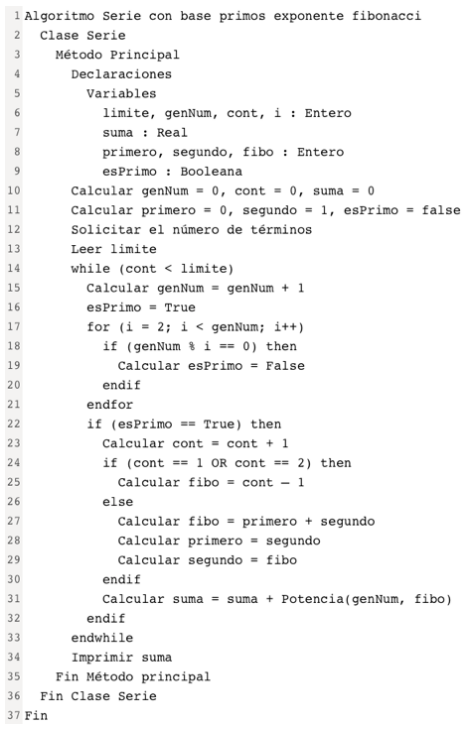
7	`Algoritmo Números perfectos`
	`  Clase TresPerfectos`
	`  Método principal()`
	`  Declaraciones`
	`  Variables`
	`  contador, generaNum, div, sumaDiv: Entero`
	`  Calcular contador = 0, generaNum = 0`
	`  while (contador < 3)`
	`  Calcular generaNum = generaNum + 1`
	`  Calcular sumaDiv = 0`
	`  for(div = 1; div < generNum; div ++)`
	`  if (generaNum Mod div == 0) then`
	`  Calcular sumaDiv = sumaDiv + div`
	`  endif`
	`  endfor`
	`  if (sumaDiv == generaNum) then`
	`  Imprimir generaNum`
	`  Calcular contador = contador + 1`
	`  endif`
	`  endwhile`
	`  Fin Método principal`
	`  Fin Clase TresPerfectos`
	`Fin`
		Este programa no interactúa con el usuario, por lo que no tiene entradas. El proceso es generar un número y determinar si es perfecto, en caso de serlo incrementar un contador, hasta que este sea igual a 3. La salida son los 3 números encontrados.

Autoevaluación 3

Pregunta	Respuesta	Retroalimentación

8	Algoritmo Suma de números primos
	  Clase SumaPrimos
	  Método principal()
	  Declaraciones
	  Variables
	  genNum, suma, contador, limite, i: Entero
	  esPrimo : Booleana
	  Calcular suma = 0, genNum = 2, contador = 0
	  Solicitar el número de elementos a sumar
	  Leer limite
	  while (contador < limite)
	  Calcular genNum = genNum + 1
	  Calcular esPrimo = True
	  for (i = 2; i < genNum; i ++)
	  if (genNum % i == 0) then
	  Calcular esPrimo = False
	  endif
	  endfor
	  if (esPrimo == True) then
	  Calcular suma = suma + genNum
	  Calcular contador = contador + 1
	  endif
	  endwhile
	  Imprimir suma
	  FIN Método principal
	  Fin Clase SumaPrimos
	Fin
		El programa recibe como entrada la cantidad de términos a generar. El proceso inicia generando un número y determinando si es primo (buscando otros divisores diferentes a 1 y el mismo número). Si el número es primo se agrega a un acumulador, caso contrario se genera otro número. La salida es la sumatoria de todos los primos.

9	Algoritmo Suma Fibonacci
	  Clase SumaFibonacci
	  Método principal
	  Declaraciones
	  Variables
	  limite, primero, segundo, fibo, suma, i: Entero
	  Solicitar límite
	  Leer límite
	  Calcular primero = 0, segundo = 1, suma = 0
	  if (limite == 1 OR limite == 2) then
	  suma = limite – 1
	  else
	  suma = suma + 1
	  for (i = 0; i < limite; i++)
	  fibo = primero + segundo
	  primero = segundo
	  segundo = fibo
	  suma = suma + fibo
	  endfor
	  endif
	  Imprimir suma
	  Fin Método principal
	  Fin Clase SumaFibonacci
	Fin
		El programa recibe como entrada la cantidad de números que se desea sumar. El proceso consiste en generar un término de Fibonacci y agregarlo a un acumulador. La salida del programa será el valor de la variable acumulador.

Autoevaluación 3		
Pregunta	**Respuesta**	**Retroalimentación**
10		La entrada es la cantidad de términos que se deben sumar. El proceso consiste en primero encontrar los términos que son números primos, para luego generar un término de Fibonacci, el siguiente paso es elevar el primo al Fibonacci y agregar ese valor a un acumulador (que debe ser de tipo Real). La salida es el valor del acumulador.

Ir a la autoevaluación

Autoevaluación 4		
Pregunta	**Respuesta**	**Retroalimentación**
1	A	Al declarar un arreglo se define el tipo de dato, lo que determina que todos los elementos de ese arreglo sean de un mismo tipo de dato.
2	C	Recuerde que los límites inferior y superior de todo arreglo son 0 y longitud de arreglo menos uno (9 en este caso) respectivamente.
3	D	Las posiciones de un arreglo son enteras, por lo tanto, la variable que represente a esas posiciones debe ser del mismo tipo entero.
4	D	Se utiliza un ciclo repetitivo por dimensión, por lo que en este caso se necesitan 4.
5	B	La longitud del arreglo siempre va entre paréntesis y el tipo de dato se ubica al final.

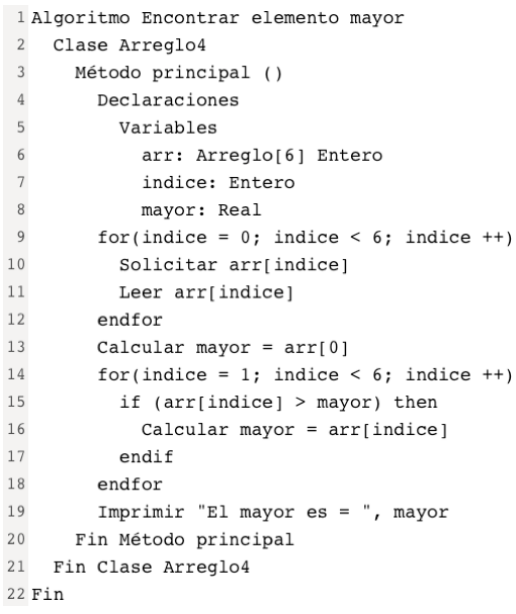
Autoevaluación 4

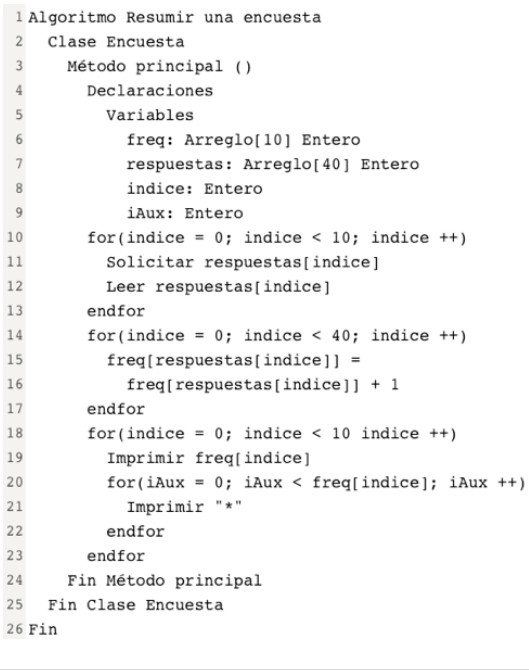
Pregunta	Respuesta	Retroalimentación

6	`Algoritmo Presenta pares de un arreglo`
	`  Clase Arreglo1`
	`  Método principal ()`
	`  Declaraciones`
	`  Variables`
	`  arr: Arreglo[7] Entero`
	`  indice: Entero`
	`  for(indice = 0; indice < 7; indice ++)`
	`  Solicitar arr[indice]`
	`  Leer arr[indice]`
	`  endfor`
	`  for(indice = 0; indice < 7; indice ++)`
	`  if (arr[indice] Mod 2 == 0) then`
	`  Imprimir arr[indice]`
	`  endif`
	`  endfor`
	`  Fin Método principal`
	`  Fin Clase Arreglo1`
	`Fin`
		El programa recibe como entradas los valores que se almacenan en el arreglo. El proceso consiste en recorrer cada elemento almacenado y determinar si es par. La salida son los números pares que se han almacenado en el arreglo.

7	`Algoritmo Presenta primos de un arreglo`
	`  Clase Arreglo2`
	`  Método principal ()`
	`  Declaraciones`
	`  Variables`
	`  arr: Arreglo[10] Entero`
	`  indice, div: Entero`
	`  esPrimo: Booleana`
	`  for(indice = 0; indice < 10; indice ++)`
	`  Solicitar arr[indice]`
	`  Leer arr[indice]`
	`  endfor`
	`  for(indice = 0; indice < 10; indice ++)`
	`  Calcular esPrimo = True`
	`  for(div = 2; div < arr[indice]; div ++)`
	`  if (arr[indice] Mod div == 0) then`
	`  Calcular esPrimo = False`
	`  endif`
	`  endfor`
	`  if (esPrimo == True) then`
	`  Imprimir arr[indice]`
	`  endif`
	`  endfor`
	`  Fin Método principal`
	`  Fin Clase Arreglo2`
	`Fin`
		El programa recibe como entrada números enteros que se almacenan en un arreglo. Para este problema, el proceso se resume en recorrer todos los elementos del arreglo y determinar si es un número primo. La salida son todos los elementos del arreglo que son números primos.

Autoevaluación 4		
Pregunta	**Respuesta**	**Retroalimentación**
8	1 Algoritmo Imprimir mayores al promedio 2 Clase Arreglo3 3 Método principal () 4 Declaraciones 5 Variables 6 arr: Arreglo[9] Entero 7 indice, suma: Entero 8 promedio: Real 9 for(indice = 0; indice < 9; indice ++) 10 Solicitar arr[indice] 11 Leer arr[indice] 12 endfor 13 Calcular suma = 0 14 for(indice = 0; indice < 9; indice ++) 15 suma = suma + arr[indice] 16 endfor 17 Calcular promedio = suma / 9 18 for(indice = 0; indice < 9; indice ++) 19 if (arr[indice] > promedio) then 20 Imprimir arr[indice] 21 endif 22 endfor 23 Fin Método principal 24 Fin Clase Arreglo3 25 Fin	La entrada del programa es un grupo de números enteros ingresados por el usuario. El proceso tiene dos fases, la primera es calcular el promedio de los elementos del arreglo y el segundo es encontrar aquellos que son mayores al promedio. La salida la constituyen todos los números que cumplen la condición mencionada anteriormente.

Autoevaluación 4		
Pregunta	**Respuesta**	**Retroalimentación**
9		Este programa necesita como entrada números reales que se almacenan en un arreglo. El proceso consiste en seleccionar como número mayor al primero elemento del arreglo, recorrer los elementos restantes y comparar cada uno de ellos con ese número seleccionado, haciendo intercambios en caso de encontrar un número más alto. La salida es el número mayor que se encontró.

Autoevaluación 4		
Pregunta	**Respuesta**	**Retroalimentación**
10		La entrada a este programa son los 40 resultados de la encuesta que se almacenan en un arreglo. El proceso consiste en poblar un arreglo que representa a las 10 posiciones de la escala y poblarlo con el número de respuesta que respondieron a la pregunta con esa escala. La salida será un histograma, construido con asteriscos; cada línea representa a una respuesta.

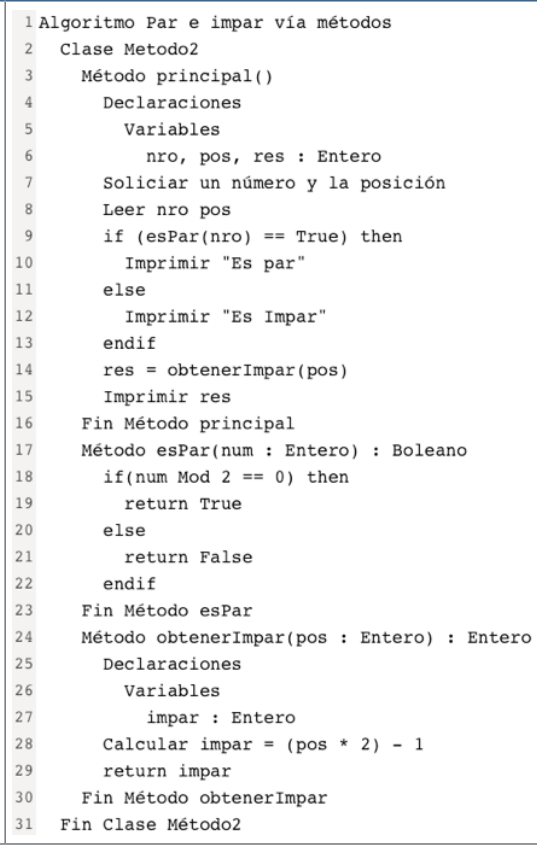
Ir a la autoevaluación

Autoevaluación 5		
Pregunta	**Respuesta**	**Retroalimentación**
1	C	Es obligatorio que un método que devuelve un valor tenga uno o varias sentencias que inician con la palabra *return.*
2	A	Los métodos que devuelven un valor en su declaración indican el tipo de dato que devolverán y lo hacen luego de la declaración de los parámetros.
3	C	El ámbito de una variable está dado por su ubicación, para que sea local debe estar dentro de un método.
4	C	Los parámetros son conceptos asociados a los métodos.

5	D	En un método pueden existir varias sentencias return. Por ejemplo:
		Método mayor(a: Entero, b: Entero): Entero
		if(a > b) {
		return a
		else
		return b
		endif
		Fin Método mayor

Autoevaluación 5		
Pregunta	**Respuesta**	**Retroalimentación**

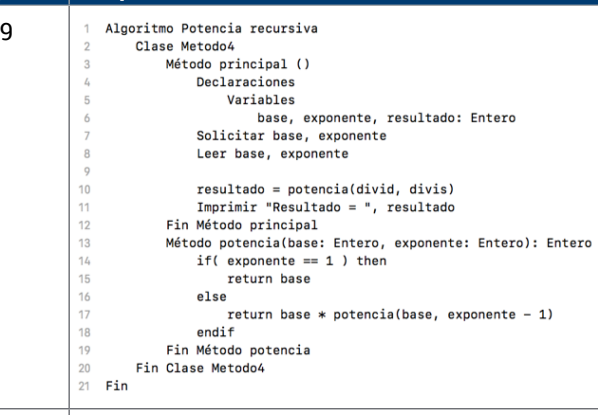
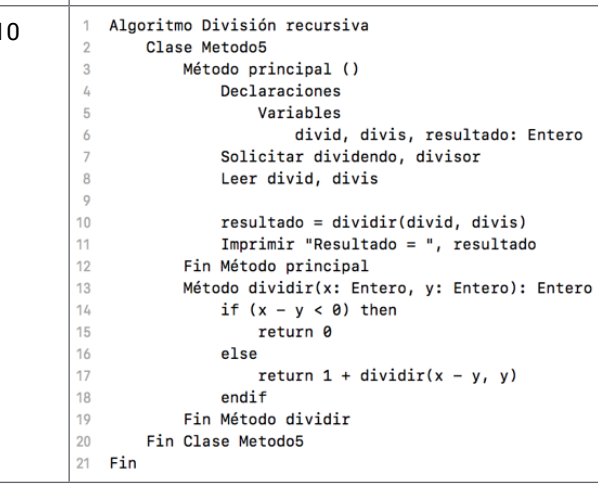
6	```
	Algoritmo Menú de métodos
	  Clase Metodo1
	  Método principal()
	  Declaraciones
	  Variables
	  opcion, res, num, b, e: Entero
	  do
	  Imprimir "1. Cálculo factorial 2. Potenciación 3. Salir"
	  Leer opcion
	  switch (opcion)
	  1 :
	  Solicitar un número
	  Leer num
	  res = calcularFactorial(num)
	  Imprimir res
	  2:
	  Solicitar base y exponente
	  Leer b, e
	  res = potencia(b, e)
	  Imprimir res
	  endswitch
	  while(opcion != 3)
	  Fin Método principal
	  Método calcularFactorial(numero : Entero) : Entero
	  Declaraciones
	  Variables
	  fac, i: Entero
	  Calcular a = 1
	  for(i = numero; i > 1; i--)
	  Calcular fac = fac * i
	  endfor
	  return fac
	  Fin Método calcularFactorial
	  Método potencia(base:Entero, exponente:Entero):Entero
	  Declaraciones
	  Variables
	  pot, i:Entero
	  Calcular pot = 1
	  for(i = 0; i < exponente; i++)
	  Calcular pot = pot * base
	  endfor
	  return pot
	  Fin Método potencia
	  Fin Clase Metodo1
	Fin
	``
		El programa muestra un menú de 3 opciones y el usuario selecciona una de ellas. Según la opción que el usuario ingresó, el programa pedirá datos adicionales de ingreso y calculará el factorial o la potencia. La salida será un número que represente al factorial o a la potencia.

Autoevaluación 5		
Pregunta	**Respuesta**	**Retroalimentación**
7		El programa tiene dos entradas: el número y la posición. Existen dos procesos, el primero es determinar si es par, para ello invoca a el método esPar, mientras que el segundo utiliza la posición encontrar el número impar aplicando una fórmula matemática sencilla. La salida es un mensaje de texto indicado si el número es o no par y el número impar que ocupa esa posición.

Autoevaluación 5

Pregunta	Respuesta	Retroalimentación

8	```
	Método esPrimo (num : Entero) : Boleano
	    Declaraciones
	        Variables
	            i : Entero
	    for(i = 2; i < num; i ++)
	        if (num % i == 0) then
	            return False
	        endif
	    endfor
	    return True
	Fin Método esPrimo
	Método sigPrimo (n : Entero) : Entero
	    Declaraciones
	        Variables
	            sPrimo : Entero
	            encontrePrimo : Boleano
	    Calcular sPrimo = n
	    while(encontrePrimo == False)
	        Calcular sPrimo = sPrimo + 1
	        if (esPrimo(sPrimo) == true) then
	            Calcular encontrePrimo = True
	        endif
	    endwhile
	    return sPrimo
	Fin Método sigPrimo
	Método obtenerElementoFibonacci (posElem : Entero) : Entero
	    Declaraciones
	        Variables
	            ant, sig, fib, i : Entero
	            ant = 0, sig = 1, fib = 0
	    if (posElem == 1) then
	        return ant
	    else
	        if(posElem == 2) then
	            return sig
	        else
	            for(i = 2; i < posElem; i ++)
	                Calcular fib = ant + sig
	                Calcular ant = sig
	                Calcular sig = fib
	            endfor
	        endif
	    endif
	Fin Método obtenerElementoFibonacci
	``
		El primer método construido se denomina *esPrimo*, recibe un parámetro que es el número que se evaluará para determinar si cumple con las condiciones que lo califican como número primo, devolviendo verdadero si lo es o falso en caso contrario. El segundo método *sigPrimo*, recibe un número entero como parámetro y devuelve otro de ese mismo tipo de dato. Este método invoca a *esPrimo*. El último método al igual que los anteriores recibe un parámetro de tipo numérico y devuelve un número que es parte de la serie de Fibonacci.

Autoevaluación 5		
Pregunta	**Respuesta**	**Retroalimentación**
9	 1 Algoritmo Potencia recursiva 2 Clase Metodo4 3 Método principal () 4 Declaraciones 5 Variables 6 base, exponente, resultado: Entero 7 Solicitar base, exponente 8 Leer base, exponente 9 10 resultado = potencia(divid, divis) 11 Imprimir "Resultado = ", resultado 12 Fin Método principal 13 Método potencia(base: Entero, exponente: Entero): Entero 14 if(exponente == 1) then 15 return base 16 else 17 return base * potencia(base, exponente - 1) 18 endif 19 Fin Método potencia 20 Fin Clase Metodo4 21 Fin	El programa recibe dos valores como entrada. El proceso principal se encuentra en el método potencia, se puede ver el caso base y el caso recursivo. La salida será un número que representa a la potencia.
10	 1 Algoritmo División recursiva 2 Clase Metodo5 3 Método principal () 4 Declaraciones 5 Variables 6 divid, divis, resultado: Entero 7 Solicitar dividendo, divisor 8 Leer divid, divis 9 10 resultado = dividir(divid, divis) 11 Imprimir "Resultado = ", resultado 12 Fin Método principal 13 Método dividir(x: Entero, y: Entero): Entero 14 if (x - y < 0) then 15 return 0 16 else 17 return 1 + dividir(x - y, y) 18 endif 19 Fin Método dividir 20 Fin Clase Metodo5 21 Fin	El programa debe recibir dos valores como entrada (dividendo y divisor). El procesamiento importante se hace en el método dividir, que consta del caso base y del caso recursivo. El resultado es un cociente de la división entera.

Ir a la autoevaluación

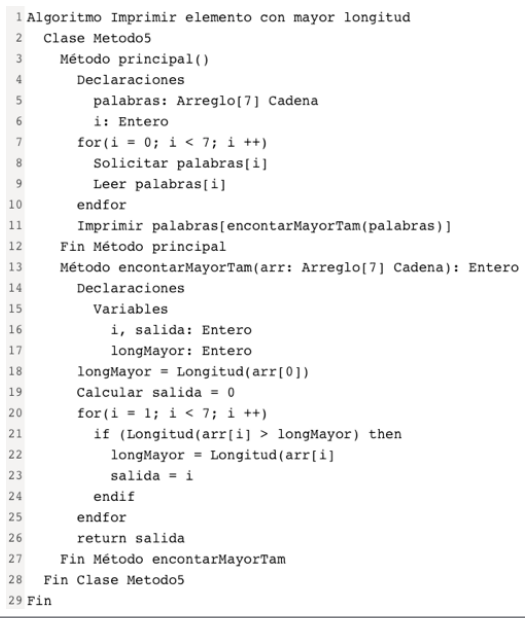
Autoevaluación 6		
Pregunta	**Respuesta**	**Retroalimentación**
1	D	*Una variable declarada del tipo Cadena se puede representar como un arreglo de caracteres.*
2	B	Al ser un arreglo de caracteres, la función devuelve el tamaño del arreglo.
3	C	En este caso es necesario ir a la quinta posición del arreglo y recorrer 3 elementos para mostrarlos.
4	B	La longitud del arreglo es 20. No olvide que los espacios en blanco también son caracteres.
5	C	Un registro es una forma de representar a varios datos.
6	A Y D	Recuerde que el acceso a estos archivos se hace línea a línea, lo que los convierte en ideales cuando se debe procesar un grupo de líneas, pero cuando se trata de búsquedas son lentos porque es necesario recorrer cada línea.

<table>
<tr><th colspan="3">Autoevaluación 6</th></tr>
<tr><th>Pregunta</th><th>Respuesta</th><th>Retroalimentación</th></tr>
<tr><td>7</td><td>A</td><td>Es la forma como se abre un archivo.</td></tr>
<tr><td>8</td><td>C</td><td>Un archivo es una colección de registros que dentro contienen datos.</td></tr>
<tr><td>9</td><td>

```
Algoritmo Determinar si un número es capicúa
  Clase Metodo4
    Método principal()
      Declaraciones
        cantidad, i,j : Entero
        valor: Boolenao
      Solicitar cantidad a validar
      Leer cantidad
      valor = esCapicua(cantidad)
      if (valor == True) then
        Imprimir la cantidad es capicúa
      else
        Imprimir la cantidad no es capicúa
      endif
    Fin Método principal
    Método esCapicua(cantidad :Entero): Booleano
      Declaraciones
        palabra: Cadena
        estado: boolean
      palabra = Cadena(cantidad)
      Calcular estado = True
      for(i=0; i < Longitud(palabra); i++)
        if(SubCadena(palabra, i, 1) !=
          SubCadena(palabra, Longitud(palabra)-(i-1), 1)) then
          Calcular estado = False;
        endif
      endfor
      return estado
    Fin Método esCapicua
  Fin Clase Metodo4
Fin
```

</td><td>Este programa recibe como entrada un número. El proceso se hace a través de métodos, convirtiendo ese número a texto y haciendo uso de las funciones estándar. La salida es un mensaje que dice si el número es capicúa o no.</td></tr>
</table>

Autoevaluación 6		
Pregunta	**Respuesta**	**Retroalimentación**

10	
	``
	Algoritmo Imprimir elemento con mayor longitud
	  Clase Metodo5
	    Método principal()
	      Declaraciones
	        palabras: Arreglo[7] Cadena
	        i: Entero
	      for(i = 0; i < 7; i ++)
	        Solicitar palabras[i]
	        Leer palabras[i]
	      endfor
	      Imprimir palabras[encontarMayorTam(palabras)]
	    Fin Método principal
	    Método encontarMayorTam(arr: Arreglo[7] Cadena): Entero
	      Declaraciones
	        Variables
	          i, salida: Entero
	          longMayor: Entero
	      longMayor = Longitud(arr[0])
	      Calcular salida = 0
	      for(i = 1; i < 7; i ++)
	        if (Longitud(arr[i] > longMayor) then
	          longMayor = Longitud(arr[i]
	          salida = i
	        endif
	      endfor
	      return salida
	    Fin Método encontarMayorTam
	  Fin Clase Metodo5
	Fin
	``
		La entrada es un grupo de palabras que se almacenan en un arreglo. El procesamiento de la entrada se realiza a través de métodos utilizando las funciones estándar de las cadenas de caracteres. La salida es la palabra más larga que ingresó el usuario.

Ir a la autoevaluación

5. Referencias bibliográficas

Downey, A., Wentworth, P., Elkner, J., & Meyers, C. (2016). *How To Think Like A Computer Scientist*: Learning with Python 3.

Downey, A. B., & Mayfield, C. (2016). *Think Java: How to Think Like a Computer Scientist*. " O'Reilly Media, Inc.".

Hommel, S. (1999). *Convenciones de código para el lenguaje de programación java*. Sun Microsystems Inc.

Horstmann, C. S. (2016). *Big Java, Binder Ready Version: Early Objects*. John Wiley & Sons.

López Román, L. (2013) *Metodología de la Programación Orientada a Objetos*. México. (2da edición). México: Alfaomega Grupo Editor.

López Vargas, J. (2018) *Guía Didáctica de Fundamentos de Programación*. Loja – Ecuador: Editorial UTPL

Lopez-Vargas, J. (2019) *Java, Apuntes básicos*. Recuperado de: http://j4loxa.com/courses/java101/Java_Apuntes_Basicos.epub

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

Rosstti, M. L. (2016) *Anorexia y Bulimia Sobreponerse a tiempo*. Recuperado de: http://bulimarexia.com.ar/pesojusto.html.