Programación Básica 1 (JAVA)

Apunte de Teoría

Programación – Algoritmos -Clasificación

Docentes:

CONDE, Sergio
BORGEAT, Andrés
MONTEAGUDO, Juan Manuel
GOITEA, Alejandro

UNIVERSIDAD NACIONAL DE LA MATANZA

Contenido

Programación Estructurada

Presentación

Los algoritmos, como procedimiento para solucionar problemas, han existido desde muchos siglos atrás, sin embargo, se han convertido en tema de interés de estudiante y profesionales por cuanto son el fundamento de la programación de computadores.

Desarrollar un programa de computador significa: indicarle al computador, en un lenguaje que él pueda entender, todos y cada uno de los pasos que debe ejecutar para lograr el objetivo propuesto, pero antes de pensar en decirle al computador cómo hacer algo, es necesario que el programador sepa cómo hacerlo. Es allí donde entra a jugar un papel preponderante el desarrollo de algoritmos, pues antes de escribir un programa es necesario diseñar el algoritmo para solucionar el problema en cuestión.

Existen varias técnicas para representar algoritmos, las más conocidas son: pseudocódigo, diagrama de flujo y diagrama N-S. En este documento se desarrolla la metodología de diagramas N-S o también conocida como diagramas de Chapin.

Algoritmo

Se sabe que la palabra algoritmo se dio en honor del matemático persa del siglo IX, Khowârizmî. Con éste término se hace referencia a un conjunto de reglas, ordenadas de forma lógica, para desarrollar un cálculo o para solucionar un problema, ya sea de forma manual o utilizando una máquina. Actualmente es frecuente hablar de algoritmo como paso previo al desarrollo de un programa de computador.

Los algoritmos están, con mayor o menor complejidad, en todas las actividades desarrolladas por el hombre y han sido utilizados por todos, infinidad de veces, sin embargo, cuando se aborda el tema como parte de la educación formal se mitifica y se difunde el prejuicio sobre que es un tema complicado.

Desde los primeros años de escolaridad se trabaja con algoritmos, en especial en el campo de las matemáticas. Los métodos utilizados para sumar, restar, multiplicar y dividir son algoritmos que cumplen perfectamente las características de precisión, finitud, definición y eficiencia.

Para que la solución de un problema sea llevada hasta un lenguaje de programación, los pasos expresados en el algoritmo deben ser lo más detallados posible, de manera que cada uno de ellos implique una operación trivial; es decir, que los pasos no impliquen procesos que requieran de una solución algorítmica. En caso de presentarse esta situación, el algoritmo debe ser refinado, lo que equivale a desarrollar nuevamente el algoritmo para la tarea concreta a la que se hace mención.

Si el problema que se desea solucionar es muy grande o complejo, es recomendable dividirlo en tareas que se puedan abordar independientemente y que resulten más sencillas de solucionar. A esto se le llama diseño modular.

Características de un algoritmo

Un algoritmo debe tener al menos las siguientes características:

- **Ser preciso**: esto significa que las operaciones o pasos del algoritmo deben desarrollarse en un orden estricto, ya que el desarrollo de cada paso debe obedecer a un orden lógico.

- **Ser definido**. Ya que en el área de programación, el algoritmo se desarrolla como paso fundamental para desarrollar un programa, es necesario tener en cuenta que el computador solo desarrollará las tareas programadas y con los datos suministrados; es decir, no puede improvisar y tampoco se inventará o adivinará el dato que necesite para realizar un proceso. Por eso, el algoritmo debe estar plenamente definido; esto es, que cuantas veces se ejecute, el resultado depende estrictamente de los datos suministrados. Si se ejecuta con un mismo conjunto de datos de entrada, el resultado será siempre el mismo.

- **Ser finito**: Esta característica implica que el número de pasos de un algoritmo, por grande y complicado que sea el problema que soluciona, debe ser limitado. Todo algoritmo, sin importar el número de pasos que incluya, debe llegar a un final. Para hacer evidente esta característica, en la representación de un algoritmo siempre se incluyen los pasos inicio y fin.
- **Presentación formal**: para que el algoritmo sea entendido por cualquier persona interesada es necesario que se exprese en alguna de las formas comúnmente aceptadas; pues, si se describe de cualquier forma puede no ser muy útil ya que solo lo entenderá quien lo diseñó. Las formas de presentación de algoritmos son: el pseudocódigo, diagrama de flujo y diagramas de Nassi/Schneiderman, entre otras.

- **Corrección**: el algoritmo debe ser correcto, es decir debe satisfacer la necesidad o solucionar el problema para el cual fue diseñado. Para garantizar que el algoritmo logre el objetivo, es necesario ponerlo a prueba; a esto se le llama verificación o prueba de escritorio.
- **Eficiencia**: hablar de eficiencia o complejidad de un algoritmo es evaluar los recursos de cómputo que requiere para almacenar datos y para ejecutar operaciones frente al beneficio que ofrece. En cuanto menos recursos requiere será más eficiente el algoritmo.

La vida cotidiana está llena de soluciones algorítmicas, algunas de ellas son tan comunes que no se requiere pensar en los pasos que incluye la solución. La mayoría de las actividades que se realizan diariamente están compuestas por tareas más simples que se ejecutan en un orden determinado, lo cual genera un algoritmo. Por ejemplo, son tareas comunes, realizar una llamada telefónica, buscar un número en el directorio telefónico, buscar un anuncio en las páginas amarillas del directorio, preparar café, regar las plantas, poner en funcionamiento un automóvil, cambiar una llanta, entre muchas otras.

Muchos de los procedimientos utilizados para desarrollar tareas cotidianas son algorítmicos, sin embargo, esto no significa que todo lo que se hace está determinado por un algoritmo. El cumplimiento de las características mencionadas anteriormente permitirá determinar si un procedimiento es o no es algorítmico.

Una receta de cocina para preparar un plato cualquiera puede ser un algoritmo, pero también puede no serlo, dependiendo de las especificaciones. Si una de las instrucciones a desarrollar dice “aplicar sal al gusto” ya puede afirmarse que no es un algoritmo, porque contiene un elemento subjetivo, no definido. Esta misma acción podría aparecer de la forma “aplicar 20 gramos de sal”, en cuyo caso no se requiere del gusto (subjetivo) de quien lo aplica y por tanto no contradice el principio de la algoritmia.

El primer paso en el diseño de un algoritmo es conocer la temática a tratar, el segundo será pensar en las actividades a realizar y el orden en que deben ejecutarse para lograr el objetivo, el tercero y no menos importante es la presentación formal.

Diagrama de Flujo

¿Qué es?: Un diagrama de flujo, se lo puede encontrar con el nombre de Flujograma de Procesos o Diagrama de Procesos, representa la secuencia o los pasos lógicos (ordenados) para realizar una tarea mediante un conjunto de símbolos. Dentro de los símbolos se escriben los pasos a seguir. Un diagrama de flujo debe proporcionar una información clara, ordenada y concisa de todos los pasos a seguir.

Se utilizan para representar algoritmos (secuencia de paso para representar una tarea). Estos diagramas nos permiten comprender una tarea y abordar a una solución generar que luego puede ser llevada a cualquier lenguaje de programación que se desee.

Podemos usar otros tipos de representación o diagramas, a modo de ejemplo nombremos los diagramas Nassi-Scheiderman que nos permiten realizar diagramación estructurada (otra forma de representación). También se puede usar el pseudocódigo de amplia difusión no solo en la Web, también algunas bibliografías se basan en esta representación para dar explicaciones del tipo teórica, sin tener que utilizar ningún lenguaje de programación específico.

Veamos a continuación las diferentes estructuras de representación:

Representación gráfica

Secuenciales

En esta parte de la representación nuestro algoritmo solo puede ejecutar una instrucción debajo de otra. Veamos los símbolos que se pueden utilizar:

Con este símbolo empiezan todos nuestros diagramas

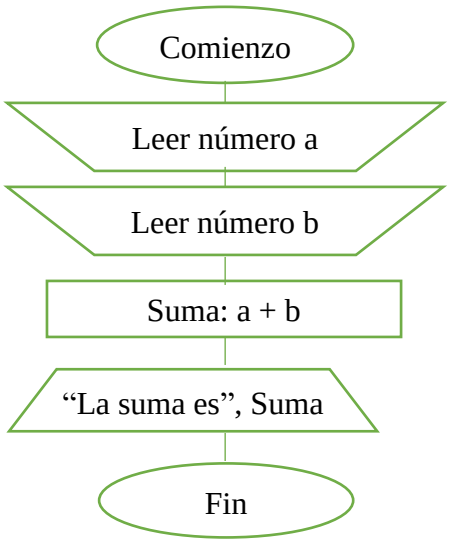
Con este símbolo se pide la entrada de información

Con este símbolo se realizan las operaciones o

Con este símbolo se muestra la salida de información

Con este símbolo finalizan todos nuestros diagramas

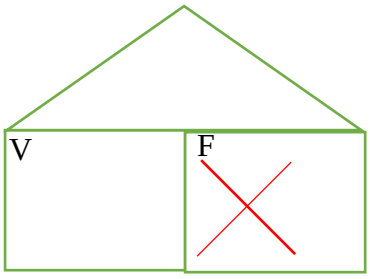
Veamos un ejemplo de la simbología mostrada, el enunciado sería dado dos números ingresar los mismo por teclado, realizar la operación de suma y luego informar el resultado.



Alternativa

- ✓ **Simple**

En este caso al realizar una pregunta del tipo condicional solo se modela una de las dos opciones.

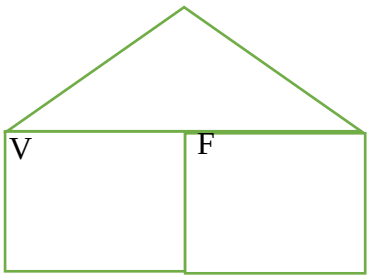


En el triángulo va la prueba lógica o condición

En la V se escribe la/s acción/es que son verdaderas

En este caso no se escribe nada en la parte falsa

- ✓ **Doble**

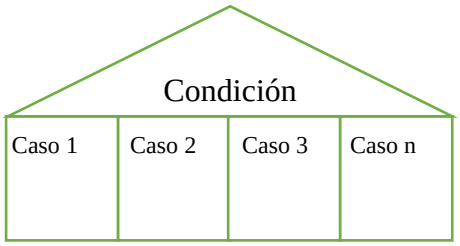


En la V se escribe la/s acción/es que son verdaderas

En la F se escribe la/s acción/es que son falsas

- ✓ **Múltiple**

En este caso por medio del ingreso de un número o carácter se accede a una de las opciones deseadas en donde se van a ejecutar una o más acciones. De ser necesario se modela una salida por defecto.

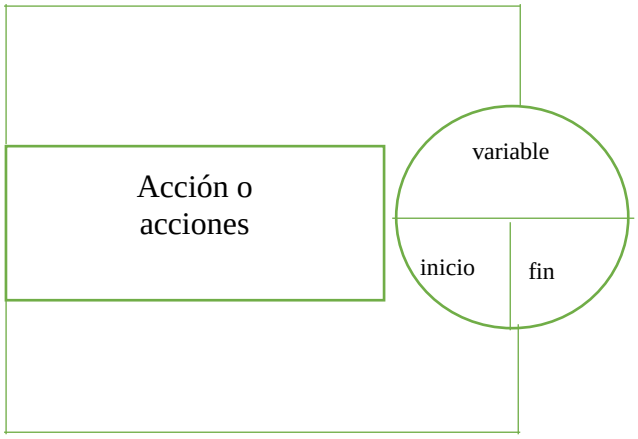


En cada caso se colocan la o las acciones a ejecutar

Ciclos repetitivos

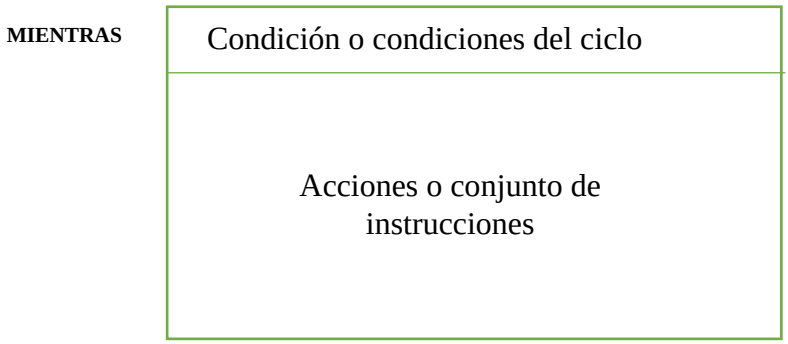
- ✓ **Ciclo con un número determinado de iteraciones**

Este ciclo se caracteriza por repetir los ciclos o iteraciones una cantidad exacta de veces producto de la condición que tiene. Se lo encuentra en la bibliografía con el nombre de "PARA". Posee una condición de inicio, una condición de fin y el incremento o decremento según se requiera.



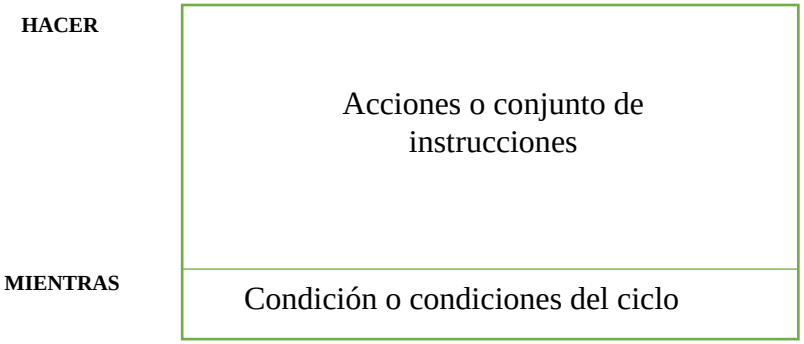
- ✓ **Ciclo con un número indeterminado de iteraciones (evalúa al inicio)**

Este ciclo se caracteriza por que las iteraciones que ocurren son manejadas con una condición de fin (puede ser una o más condiciones concatenadas). En la bibliografía puede ser que la encuentren con el nombre de "MIENTRAS". La particularidad de este ciclo es que si la condición inicial del mismo no se cumple directamente el ciclo no se ejecuta, o sea no entra al ciclo.



- ✓ **Ciclo con un número indeterminado de iteraciones (evalúa al final)**

Este ciclo se caracteriza por que las iteraciones que ocurren son manejadas con una condición de fin (puede ser una o más condiciones concatenadas). En la bibliografía puede ser que la encuentren con el nombre de "HACER-MIENTRAS". La particularidad de este ciclo es que la condición a evaluar el ciclo repetitivo está escrita al final del mismo, por lo tanto, este se ejecuta al menos una vez, en cuanto la expresión deje de cumplirse el ciclo concluye.



Representación de variables

Cuando representamos datos, numéricos o alfanuméricos, debemos darles un nombre. Una variable es un nombre que representa el valor de un dato. En esencia, una variable es una zona o posición de memoria en la computadora donde se almacena información. En un pseudocódigo y también en un programa se pueden crear tantas variables como queramos. Ejemplos:

A = 50; Variable tipo numérica A cuyo valor es 50.

Ciudad = "Asunción"; Variable alfanumérica o de tipo carácter Ciudad, cuyo valor es "Asunción"

X = C + B; Variable numérica X cuyo valor es la suma de los valores de las variables numéricas C y B. (Nota: C y B deben tener algún valor).

Asignación

Para asignar un valor a una variable se utilizará el símbolo = que en este contexto significa "es reemplazado por". De forma general tendremos que:

Nombre_variable = expresión

El valor de Nombre_variable se sustituye por el valor de expresión

Ejemplos:

C = 13
B = 25
X = C + B (después de estas instrucciones, X contendrá el valor 13 + 25 = 38)
Existe otra forma de asignar un valor a una variable, a través de la instrucción leer:

Leer (Nombre_variable)

Esta instrucción pide al usuario un valor que será asignado a la variable **Nombre_variable,** es decir, en **Nombre_variable** se almacena el valor ingresado por el usuario.

Operadores de relación

Los operadores relacionales se utilizan para formar expresiones que al ser evaluadas producen un **valor** de tipo **lógico**: **verdadero o falso.**

Signo	Operador
>	Mayor que
<	Menor que
==	Igual a
<=	Menor o igual que
>=	Mayor o igual que
<>	Distinto

Ejemplo	Resultado
25 <= 25	Verdadero
25 <> 25	Falso
25 <> 4	Verdadero
50 <= 100	Verdadero
500 >= 1	Verdadero
1 = 6	Falso

Operadores aritméticos

Signo	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
^	Potenciación
MOD	Resto de la división entera

Pseudocódigo

Instrucción Leer

La instrucción LEER se utiliza para enviar información desde un dispositivo de entrada de datos hacia la memoria. En la memoria los datos son ubicados mediante el identificador (nombre de variable) utilizado como complemento de la instrucción LEER.

En diagrama N-S la instrucción de entrada se representa así:

Leer <lista de identificadores de variables>

Ejemplo:

Leer a, b

Donde “a” y “b” son las variables que recibirán los valores y que deben haberse declarado previamente.

Instrucción Escribir

Esta instrucción permite enviar datos desde la memoria hacia un dispositivo de salida como la pantalla o la impresora. La información que se envía puede ser constante o también el contenido de variables.

Escribir <lista de constantes y variables>

Ejemplo:

Escribir a, b

Cuando se escriben más de una variable es necesario separarlas con comas (,) y los mensajes se escriben entre comillas dobles “ “. Si una variable es escrita entre comillas se mostrará el identificador y no el contenido.

Estructuras de selección

El formato de la estructura de selección es:

si (condición) entonces
instrucción 1
instrucción 2
.....
instrucción n

si no

instrucción a
instrucción b
.....
instrucción z

fin si

El resultado de evaluar la condición puede ser **verdadero o falso**: en el caso de que sea **verdadero**, se ejecutarán: instrucción 1, instrucción 2, ... , instrucción n. Si el resultado de evaluar condición es **falso** se ejecutarán: instrucción a, instrucción b, ... , instrucción z.

Ejemplo:

Si A = 5 entonces
Imprimir("A es 5")
si no
imprimir("A no es igual a 5")
fin si

Estructuras de repetición

El formato de la estructura de repetición es:

mientras (condición)

instrucción 1
instrucción 2
.....
instrucción n

fin mientras

El resultado de evaluar condición puede ser verdadero o falso:

Mientras sea **verdadero**, se ejecutarán: instrucción 1, instrucción 2, ... , instrucción n. Estas instrucciones dejarán de ejecutarse cuando condición sea **falso**.

Ejemplo:

```
leer(contraseña)
mientras (contraseña < > "joshua")
    imprimir("La contraseña es incorrecta !")
    leer (contraseña)
fin-mientras
imprimir("Ha tecleado la contraseña correcta")
```

Nota: El bucle se repetirá mientras que contraseña <> “joshua” sea verdadero

Salida de resultados

Los resultados de nuestros algoritmos los vamos a mostrar al usuario a través de la instrucción imprimir, que tiene la siguiente forma general:

Imprimir(argumento 1, argumento 2, argumento 3, ... , argumento n)

Donde los argumentos pueden ser cadenas de caracteres entrecomilladas o variables:

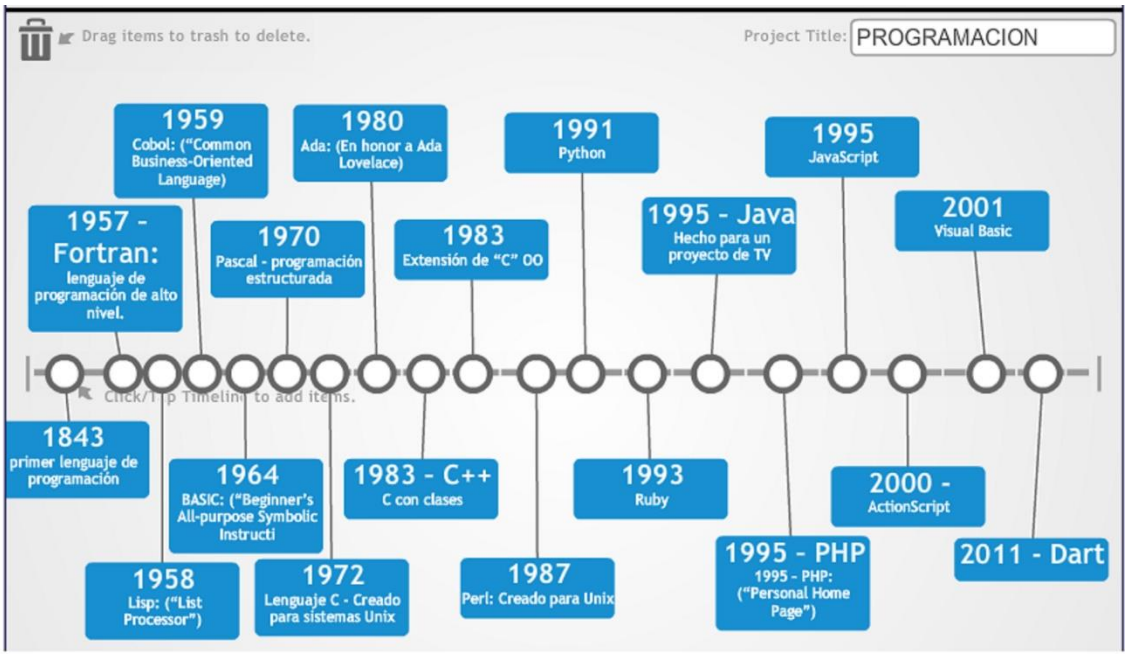
Si son cadenas de caracteres entrecomillados se imprime literalmente lo que está entre comillas

Si son variables se imprime el contenido de dicha variable (no el nombre)

Ejemplo:

sueldo = 1000
trabajador = “Martínez”
Imprimir(“el sueldo de “, trabajador , ” es “, sueldo , ” Pesos.“)
La instrucción imprimir, muestra lo siguiente: el sueldo de Martínez es 1000 Pesos.

Lenguajes de Programación



Los lenguajes de programación surgen a partir de mediados del siglo XVIII. El primer lenguaje al que se hace referencia era la programación mediante la combinación de símbolos que realizó Ada Lovelace que se conjugó en un algoritmo para la máquina analítica que todavía no estaba construida por Charles Babbage.

Recien allá por el año 1957 surge el lenguaje de programación de alto nivel denominado FORTRAN ("The IBM Mathematical Formula Translating System"). Este lenguaje se codificaba en las tarjetas perforadas que luego se colocaban en un perfograboverificador para realizar su validación, procesamiento y posterior ejecución.

En el año 1958 aparece el lenguaje LISP considerado uno de los lenguajes más simples de programación. Se componía de 8 conceptos básicos. Estaba orientado a las matemáticas y cómputos.

Por el año1959 surge el lenguaje COBOL (lenguaje común orientado a negocios). El mismo está orientado para soportar y manejar base de datos. Tenía un grado de parametrización importante. Las declaraciones debían hacerse acorde a las especificaciones que posee dicho lenguaje. Es un lenguaje multiplataforma y tiene múltiples aplicaciones.

En el año 1964 nace el lenguaje BASIC (Código simbólico de instrucciones de propósito general para principiantes en castellano), la característica principal del mismo era su simplicidad. Este lenguaje que funciona a modo interprete línea a línea tuvo su momento de mayor expansión cuando aparecen las computadoras personales en los años 70'.

Al año 1970 nace el lenguaje PASCAL (en honor a Blas Pascal) El profesor Niklaus Wirth que lo crea lo hace con la intensión que sus alumnos puedan a aplicar los conceptos de programación a un lenguaje estructurado donde también los datos estaban estructurados, permite dividir al programa en módulos o subprogramas.

En el año 1972 es el turno del lenguaje C (del antecesor lenguaje B), el manual de referencia del lenguaje para el gran público fue el libro de Kernighan y Ritchie que permitió aprender este lenguaje estructurado de gran potencia. Originalmente nació para

un entorno UNIX. Este lenguaje es la base de múltiples lenguajes que aparecieron posteriormente tales como C#, Java, JavaScript, Perl, PHP y Python.

En el año 1980 asoma el lenguaje ADA (en honor a Ada Lovelace). Es un lenguaje orientado a objetes fuertemente tipificado. Este lenguaje es solicitado por el Departamento de Defensa de los EEUU. Los encargados de desarrollarlo fueron Jean Ichbiah de CII Honeywell Bull .

Por el año 1983 aparece una extensión del lenguaje C que recibe el nombre de C++. Este nuevo lenguaje creado por Bjarne Stroustrup es orientado a objetos, por lo que se dice que es un lenguaje multiparadigma. Este lenguaje incorpora el uso de nuevos tipos de datos, plantillas y manejo de excepciones. Hasta nuestros días se sigue usando en desarrollo de aplicaciones específicas o alguna variante del mismo.

Posteriormente en el año 1987 surge el lenguaje PERL (toma características de su antecesor lenguaje C). El entorno en el que corría era UNIX y lo creo Larry Wall. Básicamente se lo creo con el fin de realizar el procesamiento de informes, se lo conoce por su versatilidad y gran potencia.

En el año 1991 surge PYTHON creado por Guido Van Rossum (lenguaje de programación que funciona en modo intérprete). Entre sus cualidades más destacadas podemos expresar su simplicidad, versatilidad y su velocidad de desarrollo. Es un lenguaje que se independiza de su plataforma y es orientado a objetos.

Dos años después en el 1993 aparece RUBY (lenguaje de programación interpretado, reflexivo y orientado a objetos), Su creador fue el japonés Yukihiro "Matz" Matsumoto. El mismo combina y se apoya en sintaxis de PYTHON y PERL. Posee una producción de código definida y agradable. Se ve influenciado por múltiples lenguajes entre los que podemos nombrar Perl, Ada, Lisp, Smalltalk, etc.

En el año 1995 ve la luz JAVA (desarrollado por fue originalmente desarrollado por James Gosling, de Sun Microsystems; posteriormente adquirido por ORACLE). Es un lenguaje de programación concurrente orientado a objetos. Nació con la necesidad de satisfacer una señal televisiva. En la actualidad tiene una amplia utilización en múltiples entornos de desarrollos y aplicaciones, para citar alguna de ellas podemos decir que cualquier sitio WEB que invoquemos seguramente se va a pedir que invoquemos a este lenguaje. Se va a utilizar en el presente curso.

Luego en el 1995 nace PHP (Procesador de Hipertexto desarrollado por Rasmus Lerdorf), un lenguaje orientado a soportar una base de datos, uso, implementación y mantenimiento de la misma. Es de código abierto. Va a ser usado del lado del servidor y se usa para el desarrollo web de contenido dinámico.

Este mismo año aparece JAVASCRPT creado por Brendan Eich (lenguaje orientado a objetos, basado en prototipos, imperativo, débilmente tipificado y dinámico). Se abrevia JS y en la actualidad todas las páginas web interpretan este código. Normalmente es utilizado del lado del cliente.

En el año 2000 es el turno del ACTION SCRIPT (su creador es la empresa ADOBE Flash). Es utilizado por los desarrolladores que programan en forma interactiva. El usuario ejecuta una película o bien un jueguito con el que puede interactuar. Está basado en especificaciones de estándar de industria ECMA-262 como JavaScript.

El VISUAL BASIC nace 2001 (en rigor aparece en 1991 y es un leguaje de definición visual que tiene su predecesor en el Basic). Su creador fue Alan Couper para Microsoft. Es un lenguaje visual y orientado por eventos, muchos lo consideran un hibrido, a pesar

de ser un lenguaje orientado a objetos no es puro. Y la empresa lo ha implementado en múltiples aplicaciones dentro de sus desarrollos. Su soporte es el paquete MSN muy conocido.

En el 2011asoma el lenguaje DART originalmente DASH (lenguaje de programación abierto desarrollado por Google). Entre las características a citar enunciamos que es de código abierto, estructurado y flexible, orientado a objetos, basado en clases, con herencia simple y soporte de interfaces, clases abstractas y tipificado opcional de datos. Sale a competir con Java Script.

PARADIGMAS DE LA PROGRAMACIÓN

Un paradigma es una propuesta o un conjunto de técnicas que se adoptan en un momento determinado. En el caso de la programación es la metodología que se va a usar propuesta por la comunidad de programadores, por la tendencia del mercado o por razones más del tipo de conveniencia o postura.

Lenguaje de máquina

En un principio, con las primeras computadoras, el programador, poseía un único lenguaje con el que interactuaba con la computadora, era el lenguaje de máquina. El mismo se apoya en el sistema binario (1 verdadero ó 0 falso).

También se utilizó otro sistema de numeración muy cercano al binario para programar, es el sistema hexadecimal que se conforma con los números del 0 al 9 y las letras A,B,C,D,E y F conformado los 16 valores posibles dentro del sistema.

DECIMAL	HEXADECIMAL	BINARIO
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	1110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Lenguaje ensamblador o Assembler

Lo complejo de programar de esta manera era que solo se podían hacer programas sencillo y de escaso tamaño.

El lenguaje assembler lo que se hacía era generar un programa fuente, que luego se va a convertir en programa objeto, este es el que se va a transformar por medio del enlace (linkeado) en el programa ejecutable. Este es el que se carga en memoria y se ejecuta.

Veamos un ejemplo de un pequeño programita en assembler:

Código objeto	Código fuente	Comentarios
01 0160	LOAD, (0160H)	Carga el contenido de la dirección del acumulador
05 0161	ADD, (0161H)	Sumar el contenido de la dirección con el acumulador
02 0162	STORE, (0162H)	Almacenar el contenido del acumulador en una dirección de memoria

En programación es la clasificación de las diferentes propuestas del modo de llevar a cabo un proyecto o un software determinado.

Existen diferentes clasificaciones de los lenguajes de programación, veamos algunas de ellas:

Clasificación de los lenguajes de programación según su Nivel de Abstracción:

1. **Lenguajes de Bajo Nivel** son Lenguajes de código máquina, son un conjunto de instrucciones en código binario que el ordenador es capaz de ejecutar directamente, específicos de cada tipo de procesador o CPU.

   Ejemplos: Lenguaje de máquina y lenguajes ensambladores.

2. **Lenguajes de Alto Nivel** más parecidos al lenguaje natural, cercanos al problema, nos olvidamos de la estructura interna del ordenador

   Ejemplos: C, Basic, Java.

Clasificación de los lenguajes de programación según su Propósito:

1. **Lenguajes de Propósito General** permiten la implementación de prácticamente cualquier algoritmo, el nivel de abstracción es más uniforme, proporciona razonable rendimiento.

   Ejemplos: Pascal, C, C++, Java, Delphi, Lisp, Scheme.

2. **Lenguajes de Propósito Específico** tienen por lo general un conjunto muy restringido de características y un alto nivel de abstracción para cumplir tareas específicas como el procesamiento de textos, gráficos, audio, video e ingeniería. Ejemplos: Snobol, SQL, Matlab.

Clasificación de los lenguajes de programación según su método de Ejecución:

1. **Lenguajes Compilados** traducen el código fuente del programa a código máquina o código objeto. Ejemplos: C, Pascal
2. **Lenguajes Interpretados** ejecutan línea a línea las instrucciones de un programa. Requieren del código fuente para ejecutar el programa. Ejemplos: Perl, Lisp.

Clasificación de los lenguajes de programación según su Paradigma

Programación Imperativa

Este paradigma surgió de un modo natural con la construcción de las computadoras: las máquinas están físicamente construidas para ejecutar series de instrucciones maquinas Se concibieron abstracciones de esas instrucciones o combinaciones de ellas para dar lugar a lenguajes de más alto nivel. La idea central es el estado del programa que va cambiando por las operaciones que se ejecutan. Podemos citar Fortran, Algol, Ada, Pascal, etc.

Programación Procedural

Para cumplir una tarea dada, descomponemos todo en sub-tareas o sub-rutinas de propósito específico que se cumplen sucesivamente para alcanzar la meta (procedimientos, funciones).

Esto nos permite tener más modularidad, es otra ventaja importante la re-utilización del código.

Programación Funcional

En la programación funcional, como lo indica su nombre, la entidad básica es la función. Es el estilo de programación declarativo que se apoya en una función para realizar tanto la entrada como la salida del mismo. Entre los usos que se le da está la Inteligencia Artificial y hoy en día en el campo de la medicina celular con sus enormes bases de datos en el entorno de investigación con R en el campo de la Bio Informática (NCBI National Center for Biotechnology Information).
Ejemplos de Lenguajes: Lisp, Haskell y R entre otros.

Programación Lógica

En este paradigma, se consideran principalmente predicados y reglas lógicas que alimentan una base de conocimientos. Es un estilo de programación derivado de la programación funcional. Ejecutar un programa, en este caso, es evaluar o un predicado o dar valores para que un predicado es verdadero. Básicamente está orientado a la Inteligencia Artificial o Sistemas Expertos.

Un ejemplo de leguaje sería el Prolog.

Programación Orientada a Objetos

La Programación Orientada a Objetos (POO en castellano o OOP en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. En síntesis podemos decir que vamos a tener un objeto, el que tiene atributos y características propias del mismo.

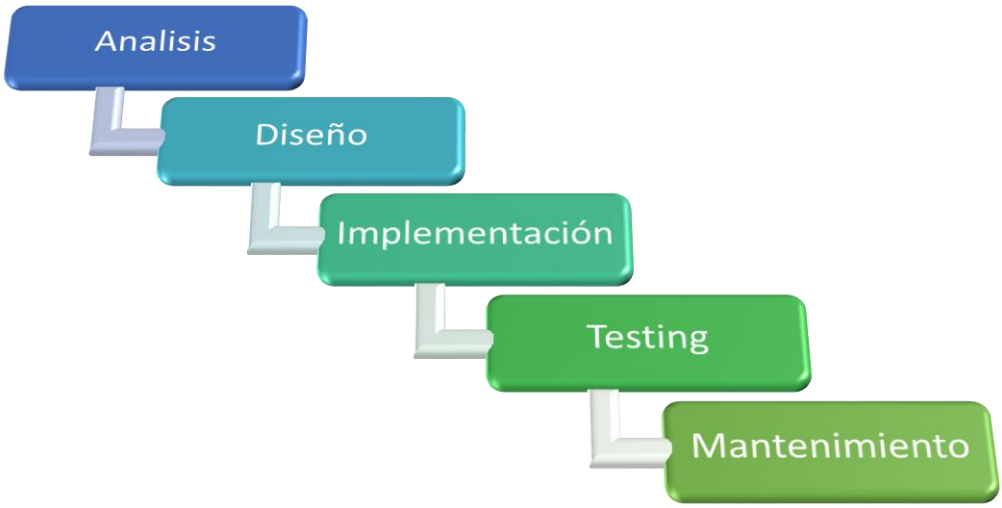
Como parte de estas características que lo engloban se pueden nombrar la herencia, modularidad, polimorfismo y encapsulamiento.

Entre los lenguajes existentes podemos citar C++, Java, Smalltalk, etc.

Pasos Para Desarrollar Un Programa

Es necesario aclarar que estos pasos o etapas son prácticamente los mismos que si se quiere desarrollar un sistema informático.

La metodología que vamos a utilizar es en cascada (se avanza en cada fase o etapa y recién cuando una concluye se pasa a la siguiente), una de las tantas que se utilizan en el entorno de la ingeniería en informática, se denomina también ciclo de vida del software.



Análisis: es cuando leemos la información que tenemos, tratamos de entender el problema en cuestión para poder empezar a construir la solución deseada.

Diseño: en este paso es en donde vamos a pensar el algoritmo que necesitamos tener para poder modelar la solución, que luego va a ser codificada en el lenguaje específico elegido.

Implementación: es la etapa donde se pasa del pseudocódigo o diagrama de flujo verificado al lenguaje de programación, se analizan las entradas, las salidas y los procesos que se desarrollan en el programa.

Testing: esta es la etapa de prueba o testeo del programa, se busca verificar que todas las salidas estén contempladas y sean las esperadas. Hay que verificar que las validaciones que contienen sean las correctas y que le brinden al usuario la información necesaria para usarlo.

Mantenimiento: En esta etapa es en donde ya el programa se encuentra funcionando, de ser necesario, se pueden realizar modificaciones o añadir nuevas funcionalidades al mismo. Los cambios pueden ser solicitados por el cliente o usuario, o bien cuando se encuentra un error que no fue detectado en alguna etapa previa.

Referencia Bibliográfica o citas Web

- ✓ Apunte de la cátedra anterior. Autores: Prof. Cintia Gioia y Prof. Carina Perez
- ✓ https://www.maestrodelacomputacion.net/historia-de-los-lenguajes-de-programacion/
- ✓ http://www.cavsi.com/
- ✓ https://asm86.wordpress.com/2009/10/12/ensamblador-desde-cero/
- ✓ http://www.portalhuarpe.com.ar/Medhime20/Sitios%20con%20Medhime/Computaci%C3%B3n/COMPUTACION/Menu/Modulo%205/5-10.htm
- ✓ http://www.areatecnologia.com/diagramas-de-flujo.htm