

Apunte parcial 1

Práctica 1

Atributos de un lenguaje de programación

- Simple y legible
- Claro en los binding
- Confiable
- Confidencial
- Tener soporte
- Abstracción
- Ortogonalidad
- Eficiencia

Por ejemplo **Python**

Ada

Lenguaje diseñado para aplicaciones de alta integridad y seguridad crítica.

- Tipos de datos: tipos escalares (enteros y reales), compuestos (arrays y registros) y tipos de punto flotante. Además admite los definidos por el usuario.
- Tipos abstractos de datos: tipos definidos por el usuario que encapsulan datos y operaciones relacionadas.
- Estructuras de datos: arrays, listas enlazadas, pilas, colas y árboles además de que el usuario puede definir otras.
- Excepciones: tiene un sistema de manejo de excepciones que permite capturar y manejarlas en tiempo de ejecución. Se definen excepciones personalizadas para sus aplicaciones y especifican cómo manejarlas según la situación.
- Concurrencia: se pueden crear programas concurrentes ejecutando múltiples tareas al mismo tiempo usando un modelo basado en procesos donde cada proceso tiene su espacio de direcciones y se comunica con otros pasándose mensajes.
- Constantes numéricas y comunes: hay dos tipos de constantes porque las numéricas refieren valores numéricos mientras que las comunes pueden contener caracteres, símbolos o cadenas de texto.

La ligación de constantes en Ada es durante la compilación.

JAVA

Lenguaje basado en C++

- Applet: programa de Java diseñado para incluirse en un documento HTML.
- Servlet: módulos de Java que permiten extender la capacidades de los servidores web.

Python, Ruby, PHP

- Python: usado para aplicaciones web, desarrollo de software, ciencia de datos y machine learning. Paradigma OO, imperativo, funcional y reflexivo.

TIPOS: acepta booleanos, numéricos y cadenas de caracteres y otros datos como list, tuple, range, set, iteradores, clases, instancias y excepciones.

- Ruby: desarrollo de aplicaciones web hasta análisis de datos. Lenguaje multiparadigma: orientado a objetos o reflexivo.

TIPOS: trata todo como un objeto.

BREAK: usado para terminar un bucle actual, normalmente usado en bucles while true. Lo bueno es que podemos definir una condición que se debe cumplir para que se ejecute el break. Otro uso potente es que puede recibir argumentos que devolverá al terminar el bucle.

- PHP: usado para crear páginas web dinámicas. Acepta imperativo, funcional, OO, procedural y reflexivo.

TIPOS: boolean, integer, float, string, array, object, resource y null.

BREAK: break finaliza la ejecución de la estructura for, foreach, while, do-while o switch en curso. Break acepta un argumento numérico opcional que indica de cuántas estructuras anidadas se debe salir. El valor predeterminado es 1.

JavaScript

JavaScript es multiparadigma porque permite la programación funcional, basada en prototipos, imperativo e interpretado. Se caracteriza por ser débilmente tipado y dinámico.

- Tipado de datos dinámico: el tipo de datos de una variable puede cambiar en tiempo de ejecución.
- Variables: las variables se pueden declarar con la palabra clave var , let o const . Las variables declaradas con var tienen un alcance de función, mientras que las variables declaradas con let o const tienen un alcance de bloque.
- Excepciones: JavaScript permite la captura de excepciones con un bloque try-catch , lo que permite manejar errores y excepciones en tiempo de ejecución.
- Funciones: lenguaje de funciones de primera clase, lo que significa que las funciones se pueden pasar como argumentos a otras funciones, se pueden asignar a variables y se pueden devolver como valores de retorno.
- Prototipado: lenguaje prototipado, lo que significa que las clases no existen como en otros lenguajes orientados a objetos. En su lugar, los objetos heredan de otros objetos, lo que permite una gran flexibilidad en la programación orientada a objetos.
- Bibliotecas y frameworks: JavaScript cuenta con una gran cantidad de bibliotecas y frameworks que facilitan el desarrollo de aplicaciones web y móviles, como React, Angular, Vue, Node.js, entre otros.

Null: representa intencionalmente un valor nulo.

Undefined: una variable a la que no se le ha asignado valor, o no se ha declarado en absoluto.


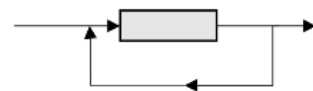

La diferencia es que null es intencionalmente declarada como vacío o nulo pero undefined no, es asignado solo por JavaScript de forma automática como valor inicial cuando se define una variable y no le da un valor.

BREAK: termina el bucle actual, sentencia switch o label y transfiere el control del programa a la siguiente sentencia seguida a la de break. La sentencia break incluye una etiqueta opcional que permite al programa salir de una sentencia etiquetada. La sentencia break necesita estar anidada dentro de la sentencia etiquetada la cual puede ser de cualquier tipo no necesariamente un bucle.

Práctica 2

BNF/ EBNF

Sistemas de notación para definir formalmente la sintaxis de los lenguajes de programación.

Meta-símbolos utilizados por		Símbolo utilizado en Diagramas sintácticos	Significado
BNF	EBNF		
Palabra terminal	Palabra terminal	Ovalo	Definición de un elemento terminal.
Digito	Digito	Rectángulo	Definición de un elemento no terminal
::=	::=	Diagrama con rectángulos, óvalos y flechas	Meta-símbolo de definición que indica que el elemento a su izquierda se puede definir según el esquema de la derecha
	()	Flecha que se divide en dos o más caminos	Meta-símbolo de opción que indica que puede elegirse uno y solo uno de los meta símbolos
< p > < p1 >	{ }		Repetición
	*		Repetición de 0 o más veces.
	+		Repetición de 1 o más veces.
	[]		Elemento optativo (puede o no estar).

Nota: p y p1 son producciones simbólicas.

La sintaxis de un lenguaje es la estructura en que se organizan los distintos elementos sintácticos como espacios, identificadores, operadores, etc. Es decir, el orden que tienen unos con respecto a otros.

La sintaxis se compone por

- Alfabeto o conjunto de caracteres.
- Identificadores.
- Operadores.
- Palabra clave y palabra reservada: las palabras claves son aquellas que tienen un significado dentro de un contexto mientras que las palabra reservadas, además de ser palabras claves, no pueden ser usadas por el programador para otros objetivos.
- Comentarios y uso de blancos.

Regla léxica: conjunto de reglas para formar las words a partir de los caracteres del alfabeto.

Regla sintáctica: reglas que definen como formar las “expresiones” y “sentencias”.

Práctica 3

Semántica

Define el significado de los símbolos, palabras y frases de un lenguaje.

Compilación

El compilador es un programa que traduce el código a lenguaje máquina para que se ejecute. El compilador toma el código de alto nivel (llamado lenguaje fuente) y después de la compilación se generará un lenguaje objeto que suele ser ejecutable.

Los compiladores tienen varias etapas

- **Análisis:** etapa en la cual interviene la semántica y es un nexo entre la etapa inicial y la final.
 - Análisis léxico: se fija que cada elemento se corresponda con alguna palabra clave transformándolo en tokens además de eliminar los comentarios, espacios en blanco, tabulaciones, etc e indica errores si alguna entrada no coincide con alguna categoría léxica.
 - Análisis semántico: analiza cada sentencia buscando estructuras, sentencias, declaraciones y esas cosas importantes y construye el árbol sintáctico del programa.
 - Análisis semántico: fase más importante donde todas las estructuras reconocidas son analizadas para hacer una comprobación de tipos.
- **Código intermedio:** código similar al objeto que surge luego de la etapa de análisis.
- **Síntesis:** construye el programa ejecutable además de optimizarlo.

Compilar no es lo mismo que interpretar el programa ya que ambos tienen ciertas diferencias.

	Ejecución	Orden	Tiempo	Eficiencia	Memoria	Errores
Compilador	Ocurre antes de ejecutar, compilando todo y generando el código objeto.	Orden físico.	Código de máquina para cada sentencia, sin repetir lazos.	Rápido en el punto de vista del HW pero la compilación es lenta.	Una sentencia puede ocupar mucho espacio.	Fáciles de ubicar y detectarlos.
Intérprete	Ejecuta sentencia a sentencia.	Orden lógico.	Para cada sentencia realiza el proceso de decodificación para determinar las operaciones y sus operandos. Si es iterativo, lo hace cuantas veces se requiera.	Más lento en ejecución.	Ocupa menos espacio porque las sentencias se transforman en un pequeño programa objeto.	Es más difícil porque el código objeto pierde las referencias del código fuente.

Errores

- Error sintáctico: error el cual incumple una regla de la sintaxis. Detectados en el proceso de compilación.
- Error semántico: incumplimiento de una regla semántica por ejemplo: división por cero, suma de string con número entero. Detectados en ejecución porque están relacionados a la lógica del programa.

Ligadura

Los programas trabajan con entidades las cuales tienen que establecerse antes de poder usar la entidad. La ligadura es la asociación entre la entidad y el atributo.

- Ligadura estática: establecida antes de la ejecución y que no podrá cambiar.
- Ligadura dinámica: si se establece en el momento de la ejecución y puede cambiarse de acuerdo con alguna regla del lenguaje. No aplica a las constantes.

Práctica 4

Inicialización

Java	Declaración e inicialización en la misma línea (int num=10;), declaración e inicialización separadas (int num; num=10;), inicialización por defecto (int num; // e inicializará en 0 automáticamente).
C	Declaración e inicialización en la misma línea (int num=10;), declaración e inicialización separadas (int num; num=10;), las variables globales y estáticas se inicializan por defecto a 0 si no se les asigna un valor explícitamente, las variables locales no tienen un valor por defecto y su valor inicial es impredecible, a menos que se les asigne un valor explícitamente.
Python	Declaración e inicialización en la misma línea (int num=10;), no es necesario declarar el tipo de variable antes de su inicialización.
Ruby	Declaración e inicialización en la misma línea (int num=10;), no es necesario declarar el tipo de variable antes de su inicialización.

Java y C tienen formas similares de inicialización de variables, mientras que Python y Ruby tienen una sintaxis más flexible y sencilla. En Python y Ruby no es necesario declarar el tipo de variable antes de su inicialización, lo que puede hacer que el código sea más fácil de escribir y entender. Por otro lado, Java y C no permiten la asignación de variables sin declarar su tipo previamente. Además, Java tiene la opción de inicialización por defecto, que es útil en ciertos casos, como cuando se desea inicializar un array.

L-valor

El atributo l-valor de una variable se refiere a la capacidad de ser referenciada o manipulada mediante una expresión de asignación. O sea que una variable con l-valor puede ser asignada a otra variable o valor

- Variable estática: ámbito de visibilidad limitado a la función o archivo en el que se declara y su valor se mantiene constante durante toda la ejecución del programa. Es decir, es una variable que existe y se inicializa en tiempo de compilación y su valor persiste en memoria durante toda la ejecución del programa.

```
def automatic_example():
    num=0
    print("El valor de num es: ",num)
    num+=1
    for i in range(5):
        automatic_example()

''' num es una variable automática dentro de automatic_example()
La salida es
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0 '''
```

- Variable automática o semi estática: declarada dentro de un bloque y su ámbito de visibilidad se limita a ese bloque. Su valor se inicializa al entrar al bloque y se destruye al salir de él. Estas variables también se llaman locales, ya que solo son visibles dentro de la función en la que se declaran.

```
def automatic_example():
    num=0
    print("El valor de num es: ",num)
    num+=1

''' num es una variable automática dentro de automatic_example()
La salida es
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0 '''
```

- Variable dinámica: variable creada y se destruye en tiempo de ejecución. En la mayoría de los lenguajes de programación, estas variables se crean usando funciones o métodos específicos como malloc() o new. El valor de una variable dinámica puede cambiar a lo largo de la ejecución del programa.

```
def dynamic_example():
    num = input ("Ingrese un número: ")
    print("El valor ingresado es:",num)

for i in range(5):
    dynamic_example()

''' num es una variable dinámica que se inicializa mediante la entrada de usuario en cada
llamada a la función dynamic_example(). La salida depende de lo que ingrese el usuario'''
```

- Variable semi dinámica: aquella cuyo tamaño puede cambiar durante la ejecución del programa, pero solo se puede ajustar al tamaño en ciertos puntos de control predefinidos. Un ejemplo de una variable semi dinámica en Ada puede ser una matriz cuyo tamaño se ajusta en tiempo de ejecución.

L-valor en C

- Variable estática: variables que se almacenan en memoria estática y mantienen su valor entre llamadas a funciones. Estas se declaran con la palabra clave "static" y se inicializan automáticamente en 0.
- Variable automática: son variables que se almacenan en la memoria de la pila y se eliminan automáticamente al salir de su ámbito de definición y su vida útil es limitada a su función o bloque de código.
- Variable dinámica: variables asignadas en tiempo de ejecución y se almacenan en la memoria dinámica. Estas variables se crean usando las funciones malloc() o calloc() y se liberan usando la función free(). Tienen una vida útil que se extiende más allá de la función o bloque de código en el que se crea.

L-valor en Ada

- Variable estática: usan la palabra constant o static y se mantienen en memoria durante toda la ejecución del programa.
- Variable automática o semi estática: definidas dentro de un bloque y se eliminan de la memoria cuando el bloque termina.
- Variable dinámica: se asignan en tiempo de ejecución usando la palabra clave new. Se mantienen en memoria hasta que se libera explícitamente usando la palabra clave delete.
- Variable semi dinámica: una matriz cuyo tamaño se puede ajustar en tiempo de ejecución pero solo en un punto de control específico del programa.

Variable local vs. variable global

- Local: variable declarada dentro de una función o bloque de código y su ámbito de alcance se limita a esa función o bloque.

Una variable local puede ser estática respecto a su l-valor, por ejemplo C o C++

- Global: variable declarada fuera de cualquier función o bloque y su ámbito de alcance es todo el programa.
 - No necesariamente las variables globales son siempre estáticas. La estática es una propiedad que puede tener una variable independientemente de su alcance. Algunos lenguajes como C, las variables globales son estática en término de su l-valor, lo que significa que su valor se mantiene durante toda la vida útil del programa y no puede ser modificado por otras funciones o bloques de código. Sin embargo, en otros lenguajes como Java y Python las variables globales no son estáticas en términos de su l-valor ya que pueden ser modificadas en tiempo de ejecución y su valor puede ser accedido y modificado por cualquier función o bloque de código en el programa.

En resumen, la naturaleza de las variables globales respecto a su l-valor no depende del hecho de que sean globales o no, sino que depende del lenguaje de programación utilizado y de cómo se manejan las variables

en ese lenguaje. Por lo tanto, es importante conocer las características del lenguaje de programación que se está usando para comprender el comportamiento de las variables globales en términos de su l-valor.

- **Variable estática respecto a su l-valor y una constante:** la diferencia principal entre una variable estática respecto a su l-valor y una constante es que el valor de una constante no puede ser modificado una vez que se ha definido, mientras que el valor de una variable estática puede ser modificado durante la ejecución del programa.

Una constante se define en el código del programa y su valor es fijo durante toda la ejecución. Es decir, el valor de una constante se define una sola vez y no se puede modificar en tiempo de ejecución.

A parte, una variable estática se define en el código del programa y su valor se mantiene en memoria durante toda la ejecución del programa, pero el valor de la variable se puede modificar en cualquier momento. Las variables estáticas se usan para mantener un valor a lo largo de la vida útil del programa.

En conclusión, la diferencia entre una estática y una constante es que el valor constante no puede modificarse después de definirse mientras que el de una estática sí puede.

Identificadores en Java

- Globales: variables estáticas de clase y pueden accederse sin necesidad de crear una instancia de la clase.
- De instancia: no estáticas, declaradas dentro de la clase y solo se acceden a través de una instancia de la clase.
- Locales: declaradas dentro de un método y solo pueden ser accedidos dentro del mismo

La diferencia entre estas 3 está basada en el alcance y duración. Las globales son visibles en toda la clase y su tiempo de vida es lo que dura la aplicación, las de instancia son específicas de cada objeto entonces duran lo mismo que el objeto exista y las variables locales solo son visibles en el método declarado y su vida se limita a la ejecución de ese bloque.

Javascript: semántica de declaración de variables

- var: el alcance es global cuando se declara fuera de una función y pueden volverse a declarar y actualizar y pueden inicializarse con un valor indefinido.
- let: si es declarada en un bloque, solo tendrá alcance en ese bloque, podrá actualizarse pero no volverse a declarar.
- const: mantienen valores constantes y si se declaran dentro de un bloque, solo podrá accederse ahí, no pueden actualizarse o volver a declarar y cada declaración debe inicializarse.