Primer Parcial

Paradigma

- Es un estilo de programación como por ejemplo la poo, secuencial, estructurada, lógica, y funcional
- Usa algoritmos para lo solución de problemas propios del proceso de construcción
- A más ejecuciones ocasiona mas procesamiento de memoria
- Los paradigmas tienen objetivos y dependen de lo que queramos hacer
- Lógica
    - Trabaja con preposiciones
    - Lógica proposicional
    - Lógica de primer orden
    - **Prolog**
        - Aprender la estructura
    - Lógica funcional
        - La programación funcional está Basada en funciones

Temas:

- Introducción e historia a los lenguajes
- POO
- Programación funcional
    - Características
    - Función
    - Expresiones
    - Tipo de datos

    - Funciones de orden superior
    - Ejercicios y proyecto LISP
- Programación lógica y orientada a aspectos
    - Características
    - Definición de Aspecto
    - Punto de corte

Historia de los lenguajes de programación

El padre de la informática moderna - Alan Turing

Las computadoras se crearon con el propósito de resolver problemas, cálculos y simular comportamiento humano o animal

Semana 2

@15 de febrero de 2024

Una función es una relación entre un conjunto de entradas y un conjunto de salidas posibles con la propiedad de que cada entrada está relacionada con exactamente una salida. En matemáticas, una función puede ser representada como una ecuación, una tabla de valores, un gráfico o en palabras.

Una función necesita una entrada y da una salida

Los paradigmas brindan la oportunidad de compartir conceptos y procedimientos

Cambios en los paradigmas importantes han provocado las revoluciones científicas

Comparten conceptos y procedimiento

Un paradigma es un modelo de trabajo compartido por una comunidad científica

El estudio de los paradigmas constituye la fuente principal en la función del estudiante, para preparar su integración a una comunidad científica particular donde llevará a cabo su practica

Los lenguajes de programación pueden estimular el uso de algunos paradigmas pero inhibir algunos.

El lenguaje de programación nació en 1801

Joseph Marie Jacquard fue el invento del telar programable conocido como Telar de Jacquard. El telar hacia el uso de tarjetas perforadas, hecha de cartón y funcionaba con el código binario

La máquina de Turing 1936

- Modelo matemático de un dispositivo que se comparta como un autómata finito y que dispone una cinta de longitud infinita en la que se pueden leer, escribir o borrar símbolos.

Fortran 1957

- Lenguaje de programación de alto nivel de propósito general, procedural e imperativo, que está especialmente adaptado al calculo numérico y a la computación científica

COBOL 1959

- Se utiliza principalmente para sistemas comerciales, bancarios y administrativos para empresas y gobiernos COBOL todavía se usa ampliamente en aplicaciones implementadas en ordenadores centrales como trabajos de lote y procesamiento de transacciones a larga escala

Basic 1957

- Medio para facilitar la programación en ordenadores a estudiantes y profesores que no fuera de ciencias

Pascal 1970

- Lenguaje de programación fuertemente tipado. esto es por parte el código esta dividido en porciones fácilmente legibles llamadas funciones o procedimientos

C 1973

- Lenguaje de tipos de datos estáticos, débilmente tipado, dispone de las estructuras típicas de los lenguajes de alto nivel y, a su vez dispone de construcciones del lenguaje que permiten un control a bajo nivel

C++ 1979

- La intención de su creación fue extender el lenguaje de programación C y añadir mecanismos que permiten el paradigma orientado a objetos
    - HTML, Python y Visual Basic - 1991
    - Java, JavaScript y PHP - 1995

Lenguaje

- El lenguaje es nuestro principal medio de comunicación , permite que nos comuniquemos entre humanos, ya sean palabras, señas o sonidos que son abstractos, tienen un significado o un sentido y señalan objetos o acciones. Así es como se logra la comunicación entre ellos.
- Gramática, semántica y pragmática
    - Gramática
        - Es esa parte de la descripción del idioma que responde a la pregunta. ¿Qué frases son correctas? una vez definido el alfabeto de una lengua como primer paso La sintaxis describe que secuencias de palabras constituyen frases legales. La sintaxis es una relación entre signos , es decir entre todas las secuencias posibles de palabras
    - La semántica
        - Dice que significa una clase correcta, es esa parte de la descripción del lenguaje que busca responder a la pregunta ¿Qué significa una frase correcta? atribuye un significado a cada frase correcta.
    - La pragmática

        - ¿Cómo usamos una oración significativa? Las oraciones con el mismo significado puede ser utilizadas de diferentes maneras por diferentes usuarios. Diferentes contextos lingüísticos pueden requerir el uso de diferentes oraciones

Lenguaje de programación

- Conjunto de instrucciones a través del cual los humanos interactúan con las computadorass. Un lenguaje de programación nos permite comunicarnos con las computadoras a través de algoritmos e instrucciones escritas en una sintaxis que la computadora entiende e insterpreta en **lenguaje máquina**

Semana 3

Lenguaje máquina

- Dada una máquina abstracta M1 el lenguaje l “entendido” por el interprete de M1 se denomina lenguaje de máquina M1

Máquina abstracta

- Suponga que se nos da un lenguaje de programación L. Una máquina abstracta para L, denotada por M1, es cualquier conjunto de estructura de datos y algoritmos que pueden realizar el almacenamiento y la ejecución de programas escritos en L
    - Una M1 es por definición un dispositivo que permite la ejecución de programas escritos en L
- **La gramática** se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas
- **La sintaxis** es la construcción de sentencias que pueden considerarse como correctos para ese lenguaje de programación. La sintaxis es la parte visible de un lenguaje de programación.
- Las reglas que determinan el significado de los programas constituyen **la semántica** de los lenguajes de programación

- Un paradigma de programación es una manera u estilo de programación de software
    - Cada lenguaje puede pertenecer a uno o varios paradigmas
- Existen diferentes formas de diseñar un leguaje de programación y varios modos de trabajar para obtener los resultados que necesitamos
- Un paradigma se trata de un conjunto de métodos sistemáticos aplicables a todos los niveles del diseño de programas para resolver problemas computacionales.

Tipos de paradigmas

Paradigma declarativo

Este paradigma no necesita definir algoritmos puesto que describe el problema en lugar de encontrar una solución al mismo. Este paradigma utiliza el principio del razonamiento lógico para responder a las preguntas p cuestiones consultadas

Se divide en 2

- Programación lógica
    - O también llamada predictiva está basada en la lógica matemática. siguiendo una serie de principios basados en hechos y suposiciones
        - Preposicional y de primer orden
- Programación funcional
    - El código de los programas funcionales está dividido en una serie de funciones, que reciben datos, operan con ellos y devuelven un valor de salida.

Paradigma imperativo

Este es un método que permite desarrollar programas a través de procedimientos. Mediante una serie de instrucciones, se explica paso por paso como funciona el código para que el proceso sea lo más claro posible

- Programación orientada a objetos: Constituye modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones. Permite separar los diferentes componentes de un programa simplificando asi su creación, depuración y posteriores mejoras
- Programación estructurada. Es un tipo de programación imperativa donde se controla el flujo utilizado condicionales, subformas y bucles (”if” o “do-while” por ejemplo) se evita utilizar los saltos absolutos entre instrucciones
- En la programación procedimental se divide el código en partes mas pequeñas y manejables llamadas procedimientos y funciones

Generaciones de los lenguajes de programación

- Primera generación: Pertenece el lenguaje máquina, consiste en secuencias de 0 y 1 y es el único lenguaje que entienden las computadoras modernas
- Segunda generación: Lenguajes ensambladores. Establecen una serie de reglas que hacen más sencilla la lectura y escritura del programa
- Tercera generación: Pertenecen los lenguajes como C, FORTRAN o Java, estos lenguajes se consideran de alto nivel ya que están bastante alejados del lenguaje máquina y son mucho más legibles por el hombre
- Cuarta generación: De propósito específico como SQL, NATURAL o ABAP. No están diseñados para programar aplicaciones complejas, sino que fueron diseñados para solucionar problemas muy concretos
- Quinta generación: Son utilizados principalmente en el área de la IA. Se trata de lenguajes que permiten especificar restricciones que se le indican al sistema, que resuelve un determinado problema sujeto a estas restricciones. Algunos son Prolog o Mercury

El propósito general de un lenguaje de programación es ayudar al programador en la práctica de su arte

---

@23 de febrero de 2024

Exposiciones

SQL

creado en 1979

Declarativo - Nivel alto

En 1999 uso paradigma orientado a objeto

Funciona con el álgebra lineal

Usado para todo tipo de servicios

C

UNIX

Paradigma estructurado por la programación modular para posteriormente enlazado

Lenguaje estructurado

Multipropósito para SO, Videojuegos, lenguaje rico con muchos tipos de datos, operadores y variables

Carencias: No tiene recolección de basura, no hay poo

Usos comunes

SO Windows MacOS, Desarrollo de software,

HTML

Lenguaje de marcado

1990 gracias por compartir archivos en la www

Se estandarizó en 1993

Paradigma

- Procedimental

Usos comunes

- Para páguinas web en su estructura

Caracteristiccas

- Mejora el contenido de una mpaguina, compatible por muchos dispositivos, facil de aprender, incorrpora multimedia

Lisp

Procesador de listas

Surgió a finales de 1950 por IA, para solventar necesidades de IA, steven roserng Dani erots

Tiene recolector de basura

Multiparadigma funcional, imperativo en programación estructurada, algoritmos recursivos, manejo de memoria automatica

Para el desarrollo de IA, Grammatly, para educación de maestría de IA

Java

En 1991 Star 7

1995 OAT

1996 Java

2007 java de cuenta abierta, 2009 la empresa zoom pasó a manos de oracle

Paradigma imperativo → poo, estruturas secuenciales, simple, multihilo, seguro, independiente de plataforma, usos comunes → para desarrollo de videojuegos, gpu de celulares, IA e Internet de las cosas.

Rust

Graydon Horare en 2010 en la empresa Mozilla

Desarrollado para correjir errores causados por C++ como la memoria inválida

Paradigma → poo, procedimental, funcional, multiparadigma

Seguro, concurrente y de buen rendimiento, confiable, fácil de usar, eficiente

Usos→ Web, servidores, videojuegos, nube, comandos, sistemas de tiempo real, frameworks.

Software → firefox, brave, Redox OS, Rust itself, Rocket, Revy, Ripgreb

COBOL

Para las aplicaciones comerciales, lenguaje común orientado a negocios

1959

Paradigma imperativo por las secuencias y procedural por las funciones
Procesamiento de datos comerciales, especifico a aplicaciones comercia;
1950 Common Business Oriented Language.
Inicio en 1959 para estandarizar procesos comerciales.
Comité compuesto por fabricas de computadora RCA, IBM y agentes

gubernamentales (fuerza área de EUA). William Selden, Gertrude Tierney.
1960 estandar de especificación.
Paradigma imperativo y procedural. En menor medida el paradigma orientado a objetos y declarativo.
Características:

- Estandarizado
- Enfocado en los negocios
- Legibilidad
- Facilidad de aprendizaje
- Lenguaje estructurado
- Datos alfanuméricos, numéricos, fecha y hora.
  Usos comunes.
- Sistemas financieros
- Sistemas bancarios
- HSBC
- Citibank
- Axxa

Kotlin

Desarrollado por JET BRAINS, a partir de 2011, para desarrollo en android en 2016

Características Interoperabilidad, Necesita la Java Virtual Machine

Paradigma orientado a objetos, paradigma funcional

Para desarrollo web, desarrollo móvil, de escritorio

Completo y versátil

GO

Creado por Google, Robert Gristmer JVM, Rhon Price UNIX, C

Publicado en 2009

Características

- Programación estructurada, librería estándar, gestión de paquetes, de tipado estático, fácil de aprender, compilado, multiparadigma

Para cloud & network, Interfaces de líneas de comandos, desarrollo web, librerías

Ensamblador

1949, Diseños de instrucciones con una letra memotecnica,

Permite a los programadores acceder al hardware y modificar la memoria,

Usos: Drivers,

Paradigma

Javascript

En los 90’s surgió una competencia por usuarios de la web

Antes llamado Mocka

Scratch

Para niños y adolescentes de una forma más sencilla

Usa el paradigma de programación visual y programación estructurada

Sistema gráfico que consiste en acomodar bloques para generar sonidos, bucles, variables, etc. Para desarrollar sistemas para la lógica de programación

Basic

1963 para enseñarle a programar a personas que estaban fuera del área de la matemática,

La primera versión surgió en mayo de 1964 y se convirtió gratuito el compilador de basic. por Bill Gates El primer lenguaje para principiantes

Paradigma: Declarativo, poo, invertido

Para motor de videojuegos

Lenguaje fácil de aprender, palabras facil de recordar, lenguaje estándar en la década de los 70’s

Ada

Desarrollado en los 80’s en francia

Poo, estructurado, imperativo, lenguaje muy exacto para definir el espacio en memoria para cada variable, en sistemas bancarios, trafico aereo,

Semana 4

HTML

Lenguaje de marcado

1990 gracias por compartir archivos en la www

Se estandarizó en 1993

Paradigma

- Procedimental

Usos comunes

- Para páguinas web en su estructura

Caracteristiccas

- Mejora el contenido de una mpaguina, compatible por muchos dispositivos, facil de aprender, incorrpora multimedia

Lisp

- Procesador de listas
- Surgió a finales de 1950 por IA, para solventar necesidades de IA, steven roserng Dani erots
- Tiene recolector de basura
- Multiparadigma funcional, imperativo en programación estructurada, algoritmos recursivos, manejo de memoria automatica
- Para el desarrollo de IA, Grammatly, para educación de maestría de IA

Java

- En 1991 Star 7
- 1995 OAT
- 1996 Java
- 2007 java de cuenta abierta, 2009 la empresa zoom pasó a manos de oracle

Paradigma imperativo → poo, estruturas secuenciales, simple, multihilo, seguro, independiente de plataforma, usos comunes → para desarrollo de videojuegos, gpu de celulares, IA e Internet de las cosas.

Rust

Graydon Horare en 2010 en la empresa Mozilla

Desarrollado para correjir errores causados por C++ como la memoria inválida

Paradigma → poo, procedimental, funcional, multiparadigma

Seguro, concurrente y de buen rendimiento, confiable, fácil de usar, eficiente

Usos→ Web, servidores, videojuegos, nube, comandos, sistemas de tiempo real, frameworks.

Software → firefox, brave, Redox OS, Rust itself, Rocket, Revy, Ripgreb

COBOL

Para las aplicaciones comerciales, lenguaje común orientado a negocios

1959

Paradigma imperativo por las secuencias y procedural por las funciones
Procesamiento de datos comerciales, especifico a aplicaciones comercia;
1950 Common Business Oriented Language.
Inicio en 1959 para estandarizar procesos comerciales.
Comité compuesto por fabricas de computadora RCA, IBM y agentes gubernamentales (fuerza área de EUA). William Selden, Gertrude Tierney.
1960 estandar de especificación.
Paradigma imperativo y procedural. En menor medida el paradigma

orientado a objetos y declarativo.
Características:

- Estandarizado
- Enfocado en los negocios
- Legibilidad
- Facilidad de aprendizaje
- Lenguaje estructurado
- Datos alfanuméricos, numéricos, fecha y hora.
  Usos comunes.
- Sistemas financieros
- Sistemas bancarios
- HSBC
- Citibank
- Axxa

Kotlin

Desarrollado por JET BRAINS, a partir de 2011, para desarrollo en android en 2016

Características Interoperabilidad, Necesita la Java Virtual Machine

Paradigma orientado a objetos, paradigma funcional

Para desarrollo web, desarrollo móvil, de escritorio

Completo y versátil

GO

Creado por Google, Robert Gristmer JVM, Rhon Price UNIX, C

Publicado en 2009

Características

- Programación estructurada, librería estándar, gestión de paquetes, de tipado estático, fácil de aprender, compilado, multiparadigma

Para cloud & network, Interfaces de líneas de comandos, desarrollo web, librerías

Ensamblador

1949, Diseños de instrucciones con una letra memotecnica,

Permite a los programadores acceder al hardware y modificar la memoria,

Usos: Drivers,

Paradigma

Javascript

En los 90’s surgió una competencia por usuarios de la web

Antes llamado Mocka

Scratch

Para niños y adolescentes de una forma más sencilla

Usa el paradigma de programación visual y programación estructurada

Sistema gráfico que consiste en acomodar bloques para generar sonidos, bucles, variables, etc. Para desarrollar sistemas para la lógica de programación

Basic

1963 para enseñarle a programar a personas que estaban fuera del área de la matemática,

La primera versión surgió en mayo de 1964 y se convirtió gratuito el compilador de basic. por Bill Gates El primer lenguaje para principiantes

Paradigma: Declarativo, poo, invertido

Para motor de videojuegos

Lenguaje fácil de aprender, palabras facil de recordar, lenguaje estándar en la década de los 70’s

Ada

Desarrollado en los 80’s en francia

Poo, estructurado, imperativo, lenguaje muy exacto para definir el espacio en memoria para cada variable, en sistemas bancarios, trafico aereo,

R

en 1978 en AT&T

R surge en 1993, DSL domain specific language

Caract

Manejo y almacenamiento de datos

IA, Analisis de datos en los bancos para analizar el comportamiento de las cuentas de sus clientes, predicciones, etc

Paradigma

mathlab

creado en 1970para facilitar el uso de hacer cáculos de matrices

Lenguahe de alto nivel, para analisis de datos, calculos de manera eficiente.

Ingeniería, matemáticas y ciencia.

Powershell

Noviembre de 2006 por ing de microsoft Jeffrey Snover

Automatización de tareas multiplataforma para diferentess SO y con un sistema de scripts basada en .NET

Tiene un estilo de scripting y de poo

Gestión remota de sistemas para servidores , etc

Incluye a la nube

para admin de sistemas, automatización de tareas, cinfiguración de redes

Ruby

por Yukihiro Matesumoto en 2995, dinámico, sintaxis legible y expresiva

De uso libre o código abierto

Permite crear apps web

Paradigma : poo y la programación iterativa & programación funcional

Lenguaje Interpretado- no tiene compilación, dinámico y flexible - se puede modificar en la ejecución, Legible y expresivo, multiplataforma

Para análisis de datos, desarrollo web y ´prototipado rápido, desarrollo de videojuegos, automatización de pruebas y tareas

Python

1989 por Guide Van Rossum, facil de usar, potente, versátil y productivo

Multiparadigma - poo , imperativa, procedural y procedimental

Interpretado, fácil de usar, tipado dinámico de alto nivel, multiparadigma, multiplataforma, amplia biblioteca, gran comunidad

Desarrorro web, apps, segurudad, videojuegos, IA, ciendia de datos, etc

Bash

1977 por Steven Bourne

1989 brian fox

1993 Chef Ramey,

Para reemplazar a un shel potente y versátil.

Posibilidad de modificar, crear o eliminar archivos

Integral de Unix y Linux

Multiples versiones

Expansión e historial de comandos

Permite crear funciones de comandos que se autoencadenan

Paradigma imperativo,

VisualBasic

1963

1991 desarrolló la primera versión

Paradigmas: poo, eventos y estructuradas

Fácil de usar, de aprender

Para SQL Server, desarrollo de apps y tareas

SAS

1 versión en 1966 por James Goodnitgth y Antony James Barr

Lenguaje versátil para administrar datos, minería de datos para predecir el comportamiento de los datos, resúmenes de datos.

Paradigma por procedimientos o imperativo

Paradigma declarativo

C#

Versátil, rendimiento y soporte a .NET, estandarizado por microsoft en los 2000

Anders Hejlisberg es el creador

Fuertemente tipado, gestión automática de la memoria, integración de .NET, Programación Asincrona,

Multiparadigma: imperativa, funcional, basada en eventos y la poo

Para desarrollo de aplicaciones de escritorio, desarrollo de apps y servicios backend, desarrollo de juegos en unity

PHP

Creado en 1994 por Rasmus Lerdorf

MultiParadigma, poo imperativo, funcional declarativo

Posee herencia,

De código abierto,

Prolog

1097 en Marsella, Francia. Alaín

interoperabilidad,

Paradigma -Lógico

Para IA, crear modelos, no es IA es un sistema experto que saca la información del “Experto”

Paradigma declarativo

AJAX

2005 Jesse James Barrett
Paradigmas: Asíncrono- Ejecutar 2 acciones al mismo tiempo, Orientado a eventos

Usado en el desarrollo web, del lado de servidores, apps web,

Pascal

1968-1969 Publicado en 1970 Nyklaus Wirth

Derivado de Algol-60

Compitió contra Basic

Paradigma imperativo procedimental y estructurada

De tipo estático, de tipado fuerte, multiplataforma, asignación de memoria por el programador

Desarrollo de aplicaciones de escritorio

Aplicaciones científicas y técnicas, sistemas embebidos y en controladores,

Mathematica

Desarrollado por Stephen Wolfram Lanzado en 1987

Lenguaje interactivo

Paradigma funcional, poo

De alto nivel

Bilbiotecas integradas, calculos de expresiones numéricas

Para mathlab, modelado, investigación, analisis de datos, matematicas.

Swift

Creado por apple

2010 Comienza su desarrollo,

Multiparadigma, orientado a protocolos, funcional, orientado a objetos

Reutilización de código, operaciones con cadenas avanzadas

Compatibilidad con objetive C

Alto rendimiento

Para iOS y macOS,

Smalltalk

en 1079

Paradigma poo

Programación con prototipo

Libreria estandar completa

Para desarrollo de aplicaciones empresariales, gestión de bancos , apps, desarrollo educativo, ia, investigación

.NET

Framework de desarrollo de software, web, apps, servicios web

Creado por microsoft Bill gates,

Multiparadigma - poo, Funcional, Imperativo y estruturada, Orinetada a eventos

Caract: Interoperabilidad entre lenguajes, recolección de basura, Herramientas de desarrollo, Escalabilidad y rendimiento, seguridad, administración de memoria, desarrollo web, ofrece entornos de desarrollo, interfaz gráfica.

Juegos de entretenimiento, integración de sistemas,

Fortran

Alternativa a ensamblador

Paradigma imperativo

De alto nivel,

A través del tiempo incorporó el paradigma Orientado a objetos

Orientado a la computación numérica, Tipado estático y fuerte, Soporte para programación paralela, de código abierto, análisis numérico, de datos.

Para ingeniería, apps científicas, análisis numérico

XML

- Lenguaje de marcado extensible
- Permite definir y almacenar datos
- 1996 se comenzó a desarrollar
- Paradigma orientado a objetos, programación funcional y procedural.
- Permite crear etiquetas personalizadas, estructurar datos de manera jerárquica, incluir metadatos, de estándar abierto,

- Para intercambiar datos estructurados, representar documentos, lanzamiento de datos, transformar datos de un modelo a otro.

C++

1980 por Bjarne Stroptup

Multiparadigma orientado a objetos, imperativo, facilita el encapsulamiento, la herencia, Programación procedimental, Programación genérica, Programación funcional,

Portatil con compiladores, de alto nivel, desarrollo de apps de escritoriom sistemas y sofware de bajo nivel, videojuegs, aplicaciones de sistemas integrados,

Usado en el campo de la salud

El mejor de propósito general y de propósito específico

POO

POO

- **Objeto:** Es una entidad que tiene estado, comportamiento e identidad. Es una instancia de una clase.
- **Método:** Función que realiza una acción específica. Es definido dentro de una clase.
- **Clase:** Es una plantilla para la creación de objetos. Define las propiedades y métodos que tendrán los objetos de esa clase.
- **Abstracción:** Es el proceso de ocultar los detalles de implementación y mostrar solo la funcionalidad al usuario.
- **Encapsulamiento:** Es la técnica que permite ocultar los datos de una clase del mundo exterior y acceder a ellos solo a través de métodos públicos.

- **Estado:** Es la descripción de un objeto en un momento determinado, generalmente se refiere a los valores de sus atributos.
- **Interfaz:** Es una descripción de los métodos que una clase deberá implementar.
- **Herencia:** Es un mecanismo que permite la creación de nuevas clases a partir de clases existentes, heredando sus propiedades y métodos.
- **Sobreescritura:** Es la capacidad de una clase hija de modificar el comportamiento de un método heredado de la clase padre.
- **Polimorfismo:** Es la capacidad de un objeto para tomar muchas formas. El mismo método puede tener comportamientos diferentes en diferentes clases.
- **Clases genéricas:** Son clases que pueden operar con cualquier tipo de dato.
- **Paquetes:** Son una forma de agrupar clases relacionadas.

- Los objetos son las entidades de primera clase que promueve la POO
- Un objeto es una entidad computacional independiente, que posee propiedades (atributos) y comportamientos (métodos)
- Los objetos se presentan ante los demás objetos como entidades encapsuladas que pueden ser utilizados a través de una interfaz conocida
- Los objetos interactúan mediante el envío de mensajes
- El envío del mensaje es la llamada a un método y la respuesta al mensaje es la ejecución

Los objetos tienen dos elementos elementalers

Atrbutos o propiedades: Son los daros que representan el oobjeto. Al conjunto de valores de los atributos que un objeto tiene en un momento dado se le denomina Estado

Comportamiento: Son las operaciones o métodos que un objeto puede ejecutar. Un método puede ser visto como una función.

UML

Lneguaje de modelado gráfico de sistemas de software

Tiene un apartado para POO

Clases y objetos

- Una clase es una descripción abstracta de un conjunto de objetos
- Puede verse a una clase como una plantilla, los objetos que usan esa plantilla pertenecen a esa clase y se dice que son una instancia de la clase
- El programador defina clases y a partir de las clases se crean objetos, puede haber muchos objetos de la misma clase
- Las clases no tienen estado, los objetos si tienen estado

4 Conceptos fundamentales de la POO

- **Abstracción:** Es el proceso de ocultar los detalles de implementación y mostrar solo la funcionalidad al usuario.
    - Es la definición de las características esenciales del objeto ignorando lo que no es relevante para el problema
    - Este término se relaciona también al concepto de cohesión que se tiene en ingeniería de software
    - Esta forma de ver los objetos facilita el diseño de sistemas
- **Encapsulamiento:** Es la técnica que permite ocultar los datos de una clase del mundo exterior y acceder a ellos solo a través de métodos públicos.
    - La encapsulación es una de las características fundamentales de un objeto
    - Consiste en separar los aspectos externos (interfaz) de los detalles de implementación internos
    - Los objetos interactúan entre si no necesitan conocer los detalles de un objeto, solo necesitan conocer las operaciones que ofrecen y que pueden explotar

    - Gracias a esto los objetos son independientes entre si y a la vez tienen la posibilidad de conectarse entre si a través de la interfaz
- **Herencia:** Es un mecanismo que permite la creación de nuevas clases a partir de clases existentes, heredando sus propiedades y métodos.
    - Se puede controlar que propiedades y métodos pueden ser vistos por otros objetos y cuales no
    - Al conjunto de propiedades y mp´petodos visibles para todos los objetos se le conoce como interfaz
    - Existen dos grados funcamentales de visibilidad
        - Public: las propiedades y métodos public constituyen la interfaz y pueden ser vistos
        - Foto
- **Polimorfismo:** Es la capacidad de un objeto para tomar muchas formas. El mismo método puede tener comportamientos diferentes en diferentes clases.

Semana 5

Herencia

- La herencia permite relaciones jerárquicas entre los objetos
- Permite que una clase integre automáticamente en si definición ciertas propiedades y métodos de otras clases
- UNO DE LOS PROPÓSITOS DE LA HERENCIA ES CREAR BJETOS MAS ESPECIALIZADOS
- La herencia promueve la reutilización de código y facilita el diseño
- Si la clase **B** hereda de la clase **A** se dice que **B** es una subclase de **A** o que **B** es hija de **A**
- También puede decirse que **A** es la superclase de **B** o que **A** es padre de **B**

Visibilidad

- Puede escogerse qué métodos y qué propiedades pueden ser heredadas por subclases.
- Solo los métodos y propiedades **protected** protegidos y **public** son heredadas
- Para los objetos que no heredan de la clase las propiedades y métodos protegidos son privados.

Polimorfismo

- Es una característica que permite qué valores de distintos tipos de datos sean utilizados en interfaces uniformes.
- Hay diferentes formas de explotar el polimorfismo, en este curso nos concentraremos en la **Sobrecarga y la sobreescritura**

Sobrecarga

- Es definir dos o más funciones o métodos que se llaman igual pero que regresan un tipo de datos diferente o bien reciben parámetros diferentes ya sea en tipo o en número.

Sobreescritura

- Es redifinir con la misma firma (tipo, nombre y argumentos) en una clase hija un método de la clase padre
- A pesar de que la forma es igual, el comportamiento puede ser diferente en la clase hija

Ejemplo La clase padre de A tiene definido el método: void procesas (int val)

- Es un mecanismo muy poderoso que nos permite explotar el polimorfismo en clases que tienen ancestros en común.

- Si un método recibe un objeto de una clase n específico también puede recibir (sin modificar nada) un objeto de cualquier clase descendiente.

Interfaces

- Puede entenderse a una interface como a un contrato
- El contrato establece los métodos que una clase que implementa el interface debe definir forzosamente.
- En el interface solo se definen prototipos de métodos (no tienen estado).
- Esto es muy útil desde el POV de diseño y división de trabajo
- Personas que colaboran se ponen de acuerdo en cuando a cómo consumir uy utilizar los objetos de otros, aun cuando no han sido creadas las clases

- Un objeto creado a partir de una clase que implementa a un interface se considera que es parte del tipo de dicho interface.
- Con esto es posible tener algo similar a la herencia miltiple
- Ya sea por herencia e implementación de un interface, el polimorfiamo funciona igual.
- Java hace uno extensivo de interfaces en su API para facilitar el manejo de características tales como hilos, tratamiento de eventos de GUL, ande namiento de estructuras de datos, serialización, etc.

Programación funcional

LISP

- Creado por John McCarty en 1958 Creó common lisp
- Lisp es reconocido como el primer lenguaje de programación funcional.
- Históricamente ha sidp uno de los lenguajes con mayor influencia

- Lisp fue el primer lenguaje donde se introdujeron constructores IF, llamadas recursivas, alocación dinámica de memoria, recolección de basura, programación interactiva, tipificación dinámica, compilación incremental, etc.

Recursividad

- A través de los años ha evolucionado y se ha segmentando en diversos lenguajes (Common Lisp, Schema, ELisp, etc.)
- Common Lisp es probablemente el estándar más popular de Lisp
- En realidad no se puede decir que Common Lisp es un lenguaje de programación funcional en el sentido puro del paradigma, actualmente es multiparadigma, permitiendo distintos estilos como el procedural, orien tado a objetos, orientado a aspectos, etc. (es capaz de adaptar cualquier paradigma)
- Si se deseara un curso más canónico de programación funcional se adop taría un lenguaje como Haskell o ML
- De ahora en adelante cuando se hable de Lisp realmente se estará hablando de Common Lisp.
- Common lisp es sólo un estandar
- Existen difversas implememntaciones de Common Lisp, las cuales son compatibles en los aspectos Core del Languaje

SBCL

- Es un lenguaje tanto declarativo como imperativo, siendo su parte declarativa la tradicional (definir funciones en términos recursivos)
- Es de tipificación dinámica.
- Es tanto compilado como interpretado a través de lo que se denomina como compilación incremental
- Es de evaluación ansiosa

- Históricamente es del paradigma funcional, pero actualmente es multi-paradigma
- Permite el REPL (Red Eval Programing Loop).
- Es un lenguaje muy eficiente, en muchos casos comparable a C

- Posee una característica que ningún otro lenguaje, Macros
- Los Macros de Lisp permiten tratar a los programas como si fueran datos, con esto el programador puede hacer programas que crean a otros programas, o extender los construututrs del lenguaje sin necesidad de realizar modificaciones al lenguaje (de esta forma se puede adaptar cualquier paradigma.)

Desventajas

- A través de los años se han segmentado las comunidades de Lisp, lo que trae como consecuencia comunidades pequeñas
- Comunidades pequeñas significa pocas bibliotecas Third-Party.
- La curva de aprendizaje no es trivial, sobre todo para poder explotar eficientemente características como las macros.

Sintaxis y semántica

- La sintaxis de Lisp es muy diferente a la de otros lenguajes
- Las reglas sintácticas en realidad son pocas.
- Los paréntesis se deben a que en Lisp todo se organiza mediante listas (las listas se representan con paréntesis)

Hola Mundo

```
(defun hello()
    (print "Hola mundo"))
```

S-expresions

- Son básicamente objetos de Lisp (nada que ver con la POO)
- Un S-expresion puede ser una lista o un átomo
- Las listas son delimitadas por paréntesis y pueden contener cualquier número de elementos separados por un espacio.
- Los átomos son todo los demás
- Los elementos de una lista son a su vez S-expresions
- Los comentarios (;) no son S-expresions, pero son eliminados en la generación de código

Uso de paréntestis, separar con espacios y después del paréntesis va una función

- La generación de código de Lisp es diferente a la de otros lenguajes, teniendo dos etapas básicas en lugar de solo una.
- El código fuente de Lisp es transformado en S-expresions a través del reader
- Luego las S-expresions son evaluadas por un evaluator, que determina si dichas S-expresions son formas válidas de Lisp, generando el código objeto
- Todo el código que escribimos en Lisp son S-expresions que a su vez son formas válidas de Lisp
  Eso es todo en cuanto la sintaxis
- Lo que resta es conocer las formas válidas de Lisp y los diversos átomos

---

@7 de marzo de 2024

▼ S-expressions

Son formas válidas de Lisp, generando el código objeto

```
(+ 2 2)
```

Átomos

- Números
- Cadenas
- Símbolos

Números

- 122, 2.0, -4.3 +1 1
- 2/3
- etc

Cadenas

Las cadenas de caracteres se colocan entre comillas y se evalúan a si mismas

```
"Hola mundo"
```

Variables

```
X ; El simbolo X
() ; Lista vacia
Foto
```

Paréntesis

Para indicar donde inicia y donde termina una expresión

Formas de Lisp

- Después del **reader** a convertido el texto fuente en S-expresions, las S-expresions pueden ser evaluadas como código Lisp por el evaluador

• Después de el reader a convertido el texto fuente en S-expresions, las S-expresions pueden ser evaluadas como coóigo Lisp por el evaluator.
• En realidad solo las S-expresions que son formas válidas de Lisp pueden ser evaluadas.
Hay dos formas generales de saber si un S-expresions es una forma válida de Lisp:
• Si es un átomo.
• Cualquier lista cuyo primer elemento sea un símbolo.

Evaluador de formas

El evaluador es una función que recibe una forma válida y regresa un valor, al que llamamos el valor de la forma:

La evaluación de formas de listas depende del símbolo del primer elemento. Este simbolo puede nombrar una de 3 cosas

- Función
- Operador especial
- Macro

```
(/ (- (+ 3 4) 1 (- 4 2)) = 3
```

En Lisp + es una función y (+ 2 3) es una llamada a la función Lisp evalua en 2 pasos:

- Los argumentos de la llamada son evaluados de izq a der. En este caso, los valores de los argumentos son 2 y 3
- Los valores de los argumentos son pasados a la función nombrada por el operador

Evaluación recursiva

Si alguno de los argumentos es a su vez una llamada a una función, será evaluado con las mismas reglas.

Ejemplo. Al evaluar la expresión (/ (-71) (-42)) pasa lo siguiente:

1. Lisp evalúa el primer argumento de izquierda a derecha (71). 7 es evaluado como 7 y 1 como 1. Estos valores son pasados a la función - que regresa 6.
2. El siguiente argumento (42) es evaluado. 4 es evaluado como 4 y 2 como 2. Estos valores son pasados a la función que regresa 2.
3. Los valores 6 y 2 son pasados a la función / que regresa 3.

- Si la forma es un simbolo se considera que ese simbolo representa a una variable, El valor de la forma es el valor asociado a la variable
- Si la forma es cualquier otro átomo (cadena, número, etc) su valor es el propio átomo (se evaluan en si mismos)
- Algunos simbolos se pueden evaluar en sí mismos como T y Nil, así como los símbolos jeyword (simbolos que empiezan con :)

Evaluación de la forma de llamada a la función

- Si el primer elemento de la lista es un símbolo que nombra a una función entonces la evaluación tiene que ver con invocar a dicha función.
- La regla de evaluación es la siguiente: evaluar los demás elementos de la lista como formas de Lisp (deben ser formas válidas) y pasar los valores resultantes como parámetros a la función.
- El valor de la forma de llamada a la función es el valor de retorno de la función.

Operadores especiales (quote)
• Un operador Lisp que no sigue la regla de evaluación es quote
• Es tan comúnmente usado que se introdujo en la sintaxis del lenguaje mediante el caracter' (comilla simple).
• La regla de evaluación de quote es -No evalúes nada, despliega lo que el usuario tecleó:

```
(quote (30
20
+230
```

• Lisp provee el operador quote como una forma de evitar que una expresión sea evaluada.

Operadores especiales (let)
• Un operador especial muy importante es let.
• Permiten asociar valores a símbolos (crear variables).

```
> (let ((x 1) (y 2))
(+ x y))
3
```

• Se utiliza para crear variables locales que solo existen dentro de un bloque de código específico.
• Las variables definidas dentro de un bloque let no afectarán a las vari- ables fuera de ese bloque.
• Las variables definidas con let solo son visibles y accesibles dentro del bloque let.
• Cuando el bloque let se ejecuta, las variables existen y se utilizan, pero cuando el bloque termina, las variables locales se eliminan.

Operadores especiales (let y let*)

⚫ let se utiliza para declarar varias variables locales al mismo tiempo.
• Las variables se declaran y asignan en paralelo, lo que significa que el valor de una variable no puede depender de otra variable definida en el mismo bloque.
⚫ let se utiliza para declarar variables locales de manera secuencial, lo que significa que el valor de una variable puede depender de las variables

declaradas previamente en el mismo bloque,
¿cómo evalúa Lisp la siguiente expresión? > (let* ((x 3) (y (+ x 3))) (+ xy))

Operadores especiales (setq)

- setq se utiliza para asignar un valor a una variable global. Esto significa que la variable estará disponible en todo el ámbito del programa Lisp, una vez que se le asigne un valor.
- Si la variable ya existe, setq modificará su valor. Si no existe, creará una nueva variable global con el nombre especificado.
- (setq x 10); Asigna el valor 10 a la variable global x.
- (setq x (+ x 5)); Modifica el valor de x (x ahora es 15)

10 * 5 + 30 / 6

= (+(* 7 8) 12 (- 9 3)+ 10) = 55

7 * 8 + 12 - 9 / 3 + 10

= (+(* 7 8) (- 12 (/ 9 3)) 10) = 75

(10 * 5 + 3) > (10 * 5 - 3) = T

(> (* 10 5) (+ 3))

7 * 8 + 12 - 9 / 3 + 10 - 4 * 3 / 2 = 69

@8 de marzo de 2024

Operadores especiales IF

- No todas las operaciones que hace un programa pueden ser representadas mediante funciones, ya que todos los argumentos de una función son evaluados antes de invocar a la misma
- Por ejemplo considerar el operador especial if.

- (if x (print “yes”) (print “no”))
- No se puede evaluar como una llamada a una función puesto que eso requeriría evaluar todos los miembros de la lista.

Evaluación de macros

- Las macros sirven en pocas palabras para que el programador sea capaz de generar sus propios operadores especiales y así extender la sintaxis del el lenguaje
- Una macro es una función que recibe S-expresions como argumentos y regresa una forma de Lisp (arbitrariamente compleja) llamada expansión que es después evaluada
- La expansión solo contiene funciones normales y operadores especiales del lenguaje (si la expansión generara mas llamadas a macros estas se continuarían expandiendo recursivamente)
- El propio lenguaje define algunas macros para por ejemplo realizar loops al estilo fo de otros lenguajes

- La evaluación de una macro en dos fases: El la primera se pasan los argumentos sin ser evaluados y en la segunda fase la expansión es evaluada siguiendo las reglas normales de evaluación
- Como los argumentos de la macro no son evaluados , técnicamente ni es necesario que sea formas válidas de Lisp. Cada macro asigna un significado a las S-expresions que recibe en virtud de como las utiliza para generar su expansión

- (defmacro duplicar (x) (* 2 x))
- (defmacro sumar (a b) (+ a b))

backquote(’)

- Una macro importante definida en el estándar es backquote(’)

- El comportamiento de backquote es igual al de quote, excepto que en una lisra puede decidirse si alguno de sis elementos será evaluado

- (defmacro duplicar (x) ‘(* 2 ,x))
- (defmacro sumar (a b) ‘(+ ,a ,b))

(set x 5)

‘(1 2, x x) = (1 2 5 5)

Listas

- Las listas son estructuras de datos dinámicas , esto es, su tamaño puede variar en el curso de la sesión de trabajo
- Son además estruturas de tipo secuencial, Es decir, para acceder a un elemento contenido en una lista es necesario recorrer previamente tosdos los que le preceden
- Las listas se represetan como cero o más elementos entre paréntesis.
- Los elementos pueden ser de cualquier tipo, incluidas las listas:
-

```
> '(La lisras (a b c) tiene 3 elementos)
(LA LISTA (A B C) TIENE 3 ELEMENTOS)
```

- Se puede construir listas usando el operador list que es una función, y por lo tanto, sus argumentos son evaluados:

¿que pasa su evaluamos la siguiente expresión?

>(list ‘mis(+ 4 2) “compañeros”)

- Los programas Lisp se representa como listas

- Si una lista es precedida por el operador quote, la evaluación regresa la misma lista, en otro caso, la lista es evaluada como si fuese código

```
> (list '(+ 2 3) (+ 2 3))
((+ 2 3) 5)
```

- En lisp hay 2 formas de representar la lista vacia, como un par de paréntesis o con el símbolo nil

```
Nil
```

Ejercicio Hacer una función que recibe 2 números, la función regresa el mayor de ellos

```
(defun max-num (num1 num2)
  (if (> num1 num2)
      num1
      num2))
```

Esta función, llamada "max-num", acepta dos números como argumentos y devuelve el más grande de los dos.

Hacer una función que recibe una lista x, la función debe regresar una lista de listas con 2 elementos de la forma: ((1 2 3) x)

```
(defun list-of-lists (x)
  (list '(1 2 3) x))
```

```
(defun max-num (num1 num2)
  (if (> num1 num2)
      (format t "~d es el número mayor" num1)
      (format t "~d es el número mayor" num2)))
```

En este ejemplo, se asume que `sbcl` está en tu ruta del sistema y que `maxnum.lisp` está en el directorio actual.

Semana 6

Cons

- Para construir listas

En Lisp, `cons` se utiliza para construir listas y pares. `cons` toma dos argumentos y los une en un nuevo par.

lisp Evalua si es una lista

Condicional if

- Normalmente recube 3 argumentos, una expresiónde test, una expresión then y una excepción else
- Liso evalua de la siguiente

Creación de la lista de listas

Aquí se crea una lista de listas con los nombres y los promedios correspondientes:

```
(setq lista-de-estudiantes '((ivan 9) (axel 9) (yael 9) (Maddeline 7)))
```

Para sacar el nombre de la persona con el promedio, se puede utilizar `FIRST` o `CAR`. Por ejemplo, para obtener el nombre de la primera persona en la lista:

```
(FIRST (FIRST lista-de-estudiantes)) ; devuelve 'ivan'
```

Para obtener el segundo elemento (el promedio) de la primera persona en la lista, se puede utilizar `SECOND` o `CDR`:

```
(SECOND (FIRST lista-de-estudiantes)) ; devuelve 9
```

Para crear una lista de 5 elementos y guardarla en una variable, podríamos hacer lo siguiente:

```
(setq mi-lista '(1 2 3 4 5))
```

Ahora, para crear una nueva lista que contenga el segundo y cuarto elemento de la lista creada:

```
(setq nueva-lista (list (second mi-lista) (fourth mi-list
a)))
```

Finalmente, aquí hay tres formas diferentes de mostrar el tercer elemento de `mi-lista`:

```
(third mi-lista)
(car (cddr mi-lista))
(nth 2 mi-lista)
```

@15 de marzo de 2024

dos numeros del mismo tipo con el mismo valor evalden a

Verdad, falsedad e igualdad
= para comparar números.
⚫ char= para comparar caracteres (los caracteres se representan con # seguido del numero o texto).
⚫ string para comparar cadenas (los cadenas se representan comillas).
⚫ equal: Está pensado para operar en cualquier tipo de objeto, considera a dos listas iguales si tienen la misma estructura y contenido, también considera a dos cadenas equivalentes si tienen los mismos caracteres.
⚫equalp: igual a equal sólo que en el caso de las cadenas ignora mayúscu las y minúsculas.

⚫eq: Verifica identidad, verdadero si los dos objetos son exactamente el mismo. No es recomendable utilizarlo con números ya que dependiendo de la implementación de Common Lisp su comportamiento puede variar. eql: similar a eq sólo que garantiza que domismo tipo con.
el mismo valor evalúen a T.
Linidad 3. Pogramme Final

- Sirven para abstraer funcionalidad
- Common Lisp define muchas funciones para el programador
- Siendo originalmente funcional, Common Lisp basa toda su funcionalidad en funciones (incluso las macros terminan convirtiéndose en funciones)
- Todos los tipos de datos definidos por el lenguaje dependen de que funciones pueden aplicárseles

- Es posible definir nuevas funciones con defun que toma normalmente tres argumentos:
  - Un nombre
  - Una lista de parámetros

Funciones

• El primer argumento de defun indica que el nombre de nuestra función definida será tercero.
• El segundo argumento (lst) indica que la función tiene un sólo argumento. Un símbolo usado de esta forma se conoce como variable. Cuando la variable representa el argumento de una función, se conoce como parámetro.
El resto de la definición indica lo que se debe hacer para calcular el valor de la función -Para cualquier lst, se calculará el primer elemento, del resto, del resto del parámetro (caddr lst):

```
1 > (tercero '(abcde))
```

La función `append` en Lisp se utiliza para unir dos o más listas en una sola. Toma como argumentos las listas que se quieren unir y devuelve una nueva lista que es la concatenación de estas. Aquí un ejemplo:

```
(append '(1 2 3) '(4 5 6)) ; devuelve (1 2 3 4 5 6)
```

Para programar la serie de Fibonacci en Lisp usando listas, podríamos usar una función recursiva que construye una lista con los números de Fibonacci:

```
(defun fibonacci (n)
  (if (<= n 1)
      '(0 1)
      (let ((prev (fibonacci (- n 1))))
        (append prev (list (+ (second prev) (first pre
v)))))))
```

Esta función toma un argumento `n` que es el número de términos que queremos en la serie de Fibonacci. Si `n` es 1 o menos, simplemente devolvemos una lista con los dos primeros términos de la serie de Fibonacci `(0 1)`.

Si `n` es mayor que 1, primero calculamos la serie de Fibonacci para `n - 1` términos utilizando un llamado recursivo a la función `fibonacci`. Luego, añadimos un nuevo término a la serie que es la suma de los últimos dos términos de la serie calculada hasta ahora.

Por ejemplo, para obtener los primeros 6 términos de la serie de Fibonacci, podríamos llamar a la función `fibonacci` de la siguiente manera:

```
(fibonacci 6) ; devuelve (0 1 1 2 3 5)
```

serie de Fibonacci usando recursividad y `if` para evaluar si el número a evaluar es mayor que 0:

```
(defun fibonacci (n)
  (if (<= n 0)
      0
      (if (= n 1)
          1
          (+ (fibonacci (- n 1)) (fibonacci (- n 2))))))
```

función llamada `fibonacci` que toma un argumento `n`, el número de términos a generar. Si `n` es 0 o menos, la función devuelve 0. Si `n` es 1, la función devuelve 1. Si `n` es mayor que 1, la función genera el término de Fibonacci calculando la suma de los dos términos anteriores en la serie, lo que se logra mediante llamadas recursivas a la función `fibonacci`.

Semana 7

función recursiva en Lisp que suma todos los números anteriores al número ingresado:

```
(defun sumar-hasta (n)
  (if (<= n 0)
      0
      (+ n (sumar-hasta (- n 1)))))
```

si ingresas `5`, la función devolverá `15` porque `1 + 2 + 3 + 4 + 5 = 15`.

El texto seleccionado describe una función recursiva en el lenguaje de programación Lisp que suma todos los números anteriores al número ingresado. Si el número ingresado es '5', la función sumará todos los números desde el '1' al '5' (1+2+3+4+5), devolviendo el resultado '15'. La recursión ocurre cuando la función se llama a sí misma dentro de su propia definición.

La función `sumar-hasta` toma un número `n` como input.

1. Inicialmente, la función comprueba si `n` es menor o igual a cero mediante `(<= n 0)`.
2. Si el número es menor o igual a cero, la función devuelve cero y se detiene; es decir, no hay más recursión.

3. Si `n` es mayor que cero, la función suma `n` al resultado de la llamada recursiva de `sumar-hasta` con `n-1` como argumento.
4. Esta llamada recursiva significa que `sumar-hasta` se ejecutará de nuevo, pero esta vez con `n-1`.
5. Este proceso se repetirá hasta que `n` sea cero, momento en el que la función empezará a devolver los resultados de las sumas a la llamada anterior, hasta que finalmente se devuelva el resultado final al lugar original donde se llamó a `sumar-hasta`.

---

función recursiva en Lisp que suma los cuadrados de todos los números anteriores al número ingresado:

```
(defun sumar-cuadrados-hasta (n)
  (if (<= n 0)
      0
      (+ (* n n) (sumar-cuadrados-hasta (- n 1)))))
```

si ingresas `3`, la función devolverá `14` porque `1^2 + 2^2 + 3^2 = 14`.

```
(defun calcular-Q (n)
  (/ (* n (+ n 1) (+ (* 2 n) 1)) 6))
```

Diapositivas

- Una función regresa por defecto la última forma evaluada
- El operador especial RETURN FROM puede cambiar ese comportamiento de ser necesario (aunque en la práctcia no suele ser el caso).

```
(defun acumula-lista (list)
    (when (null list)
```

```
        (return-from acumula-lista 0))
    (+ (first list)
        (acumula-lista (rest list))))
```

- (ACUMULA-LISTA '())
  0
- (ACUMULA-LISTA '(1 2 3 4))
  10

```
 (set ls '(1 2 3 4))


  (set 'ls '(1 2 3 4))

  (setf ls '(1 2 3 4))

   (setf (car ls) 10)



0] (null '())
T

0] (null nil)
T

0] (null t)
NIL

0] (null 1)
NIL
```

Funciones anónimas

- Es posible definir una función sin nombre mediante lambda
- Se pueden utilizar funciones anónimas cuando se requiere describir una funcionalidad ,uy sencilla que no será reutilizada
- A las funciones anónimas también se les conoce como expresiones lambda
- Una expresión lambda regresa un objeto función:

```
(lambda (x) (+ * 100)) 1)

 (funcall #'(lambda (x) (+ * 100)) 1)

  (lambda (x) (+ * 1))

   (funcall (lambda (x) (+ * 1)) 6)
```

@21 de marzo de 2024

función recursiva en Lisp que crea una lista desde `1` hasta el número `n` ingresado:

```
(defun lista-hasta (n)
  (if (<= n 0)
      nil
      (cons n (lista-hasta (- n 1)))))
```

Por ejemplo, si ingresas `5`, la función devolverá `(5 4 3 2 1)`.

```
(defun lista-hasta-descendente (n)
  (if (<= n 0)
      nil
      (cons (lista-hasta-descendente (- n 1)) n)))
```

Por ejemplo, si ingresas `5`, la función devolverá `(1 2 3 4 5)`.

```
(defun lista-hasta-ascendente (n)
  (if (<= n 0)
      nil
      (append (lista-hasta-ascendente (- n 1)) (list n))))
```

La función `append` en Lisp se utiliza para concatenar dos o más listas en una sola. Toma como argumentos las listas que se desean concatenar.

MAPCAR

Aplica la función para elementos de una lista