Asignatura 780014 Programación Avanzada

"Odié cada minuto de entrenamiento, pero no paraba de repetirme: 'No renuncies, sufre ahora y vive el resto de tu vida como un campeón'"

Muhammad Ali

Asignatura 780014
Programación Avanzada

TEMA 1 – PARADIGMAS DE PROGRAMACIÓN

Paradigmas de Programación

- Objetivo del tema
  - Observar los lenguajes de programación desde una perspectiva de abstracción que nos permita extrapolar los conocimientos ya adquiridos sobre otros lenguajes.

Paradigmas de Programación

- Objetivo del tema
  - Observar los lenguajes de programación desde una perspectiva de abstracción que nos permita extrapolar los conocimientos ya adquiridos sobre otros lenguajes.

- ¿Por qué?
  - En este punto:
    - Alcanzado un nivel de programación básico
    - Varios lenguajes conocidos (Python, Java, C++)
  - Hay que preguntarse:
    - ¿Sólo soy un programador o soy un ingeniero?
    - ¿Cómo de lejos estoy de ser un programador experimentado?

Índice

1. Concepto de paradigma
2. Niveles de cumplimiento
3. Características de los paradigmas
4. Algunos paradigmas
5. Ejemplos de programas
6. Ejercicios

Concepto de paradigma

- Como primera idea
  - Un paradigma de programación (PdP) es un conjunto de **características** comunes a varios lenguajes o que diferencian un cierto lenguaje de los demás
    - Una metodología no es un paradigma

http://en.wikipedia.org/wiki/Programming_paradigm

  - Nuevos lenguajes han introducido nuevos PdP
    - Fortran (1957 - imperativo)
    - Lisp (1959 - funcional)
    - Simula (1967 - orientado a objetos)
    - Prolog (1972 - lógico)

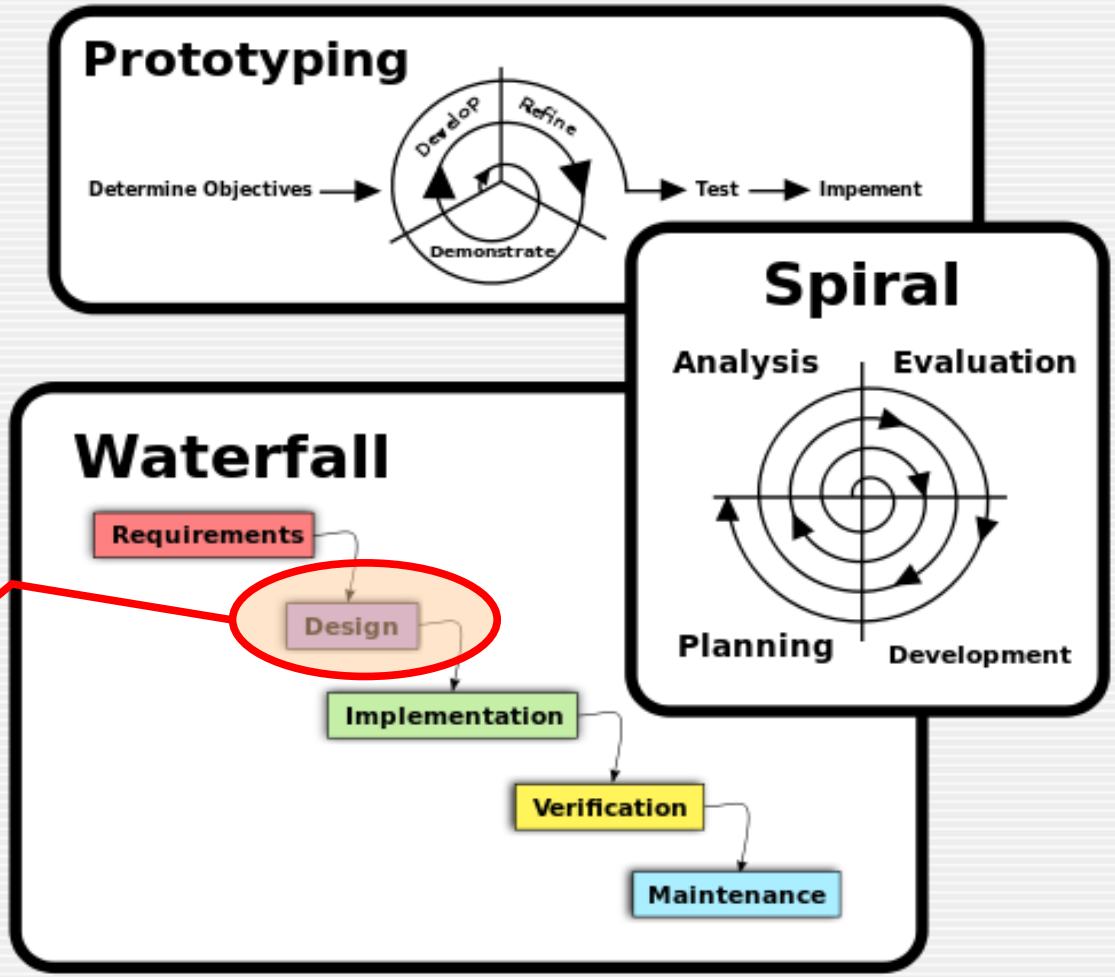
Concepto de paradigma: algunos ejemplos

- Programación concurrente
  - Programación distribuida (subtipo)
    - Programación cliente/servidor (sub-subtipo)
- Programación de tiempo real
- Programación imperativa
- Programación declarativa
  - Programación funcional (subtipo)
  - Programación lógica (subtipo)
- Programación estructurada
- Programación orientada a módulos
- Programación orientada a objetos
- Programación orientada a procedimientos

Concepto de paradigma: utilidad

- **Establecer clasificaciones**
  - Ayuda a comprender la evolución, relaciones y características de los lenguajes de programación.
- **Describir lenguajes**
  - En base a los paradigmas que cumplen.
- **Seleccionar el lenguaje para utilizar en un desarrollo**
  - Basándose en las necesidades del proyecto enunciadas como paradigmas.
- **Establecer características de un nuevo lenguaje**
  - Planificar sus características para que se ajuste a lo deseado.

Concepto de paradigma: utilidad

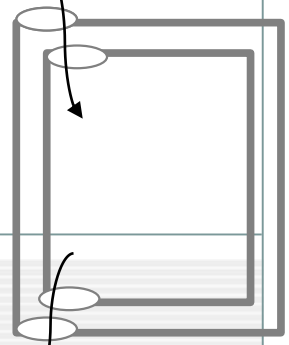


Fase donde se usan los PdP

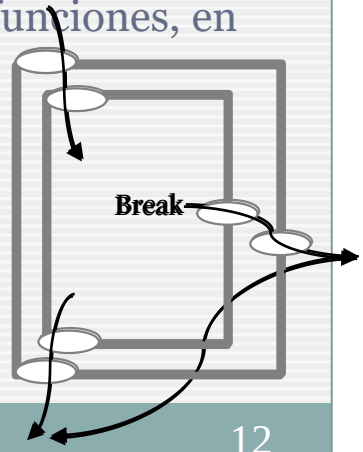
Niveles de cumplimiento de un paradigma

- **Paradigma obligatorio (siempre)**
  - Será imposible no seguir el paradigma porque no hay construcciones en el lenguaje que permitan salir de la norma que define el paradigma.
- **Paradigma soportado (fácil)**
  - Cuando es la recomendación más común para el lenguaje, cuando la programación habitual lo sigue e incluso resulta complejo y difícil no cumplir el paradigma.
- **Paradigma permitido (difícil)**
  - Es posible programar siguiendo el paradigma pero no es sencillo o cómodo porque el lenguaje no favorece su utilización.
- **Paradigma no permitido (imposible)**
  - Un PdP no puede ser utilizado o conseguido con este lenguaje porque las construcciones del mismo no lo permiten.

Niveles de cumplimiento: ejemplos

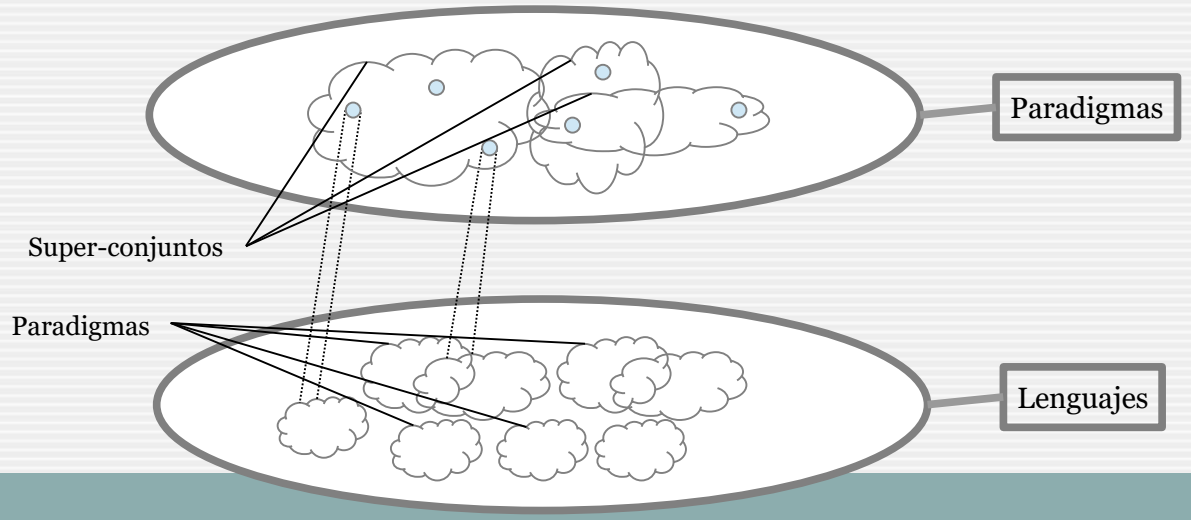


- **Pascal, y la programación estructurada**
  - Soportado: permite la programación estructurada, aunque es posible utilizar la orden "*goto*" y romper con este PdP.
- **Java, y la programación orientada a objetos**
  - Obligatorio: No es posible realizar ni un solo programa que no defina una clase y use objetos.
- **C, y la programación estructurada**
  - Permitido: "*return*" y "*break*" permiten romper los bloques de código y funciones, en contra de la programación estructurada.
- **Fortran, y la programación orientada a objetos**
  - No permitido: no existe ninguna forma de crear objetos.



Características de los PdP: relaciones

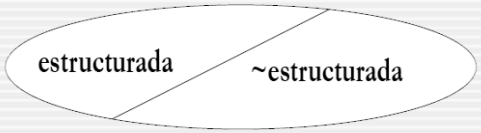
- Los paradigmas son conjuntos de lenguajes
  - También se pueden agrupar en super-conjuntos
  - Son posibles distintas distribuciones de los paradigmas en super-conjuntos
    - Según el tipo de **clasificación** de lenguajes que crean los paradigmas (2 tipos)
    - Según la **evolución** de los lenguajes y aparición de nuevos paradigmas
    - Según la **relación con el HW** de los lenguajes de un cierto paradigma



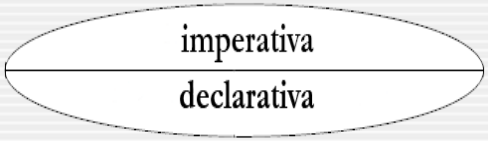
Super-conjuntos: según clasificación (tipo 1)

- Podemos agrupar paradigmas según la **clasificación** que crean sobre los lenguajes:

1. **Paradigmas generales**
   - Establecen una partición sobre el total de los lenguajes de programación
   - No cumplir el paradigma tiene relevancia
   - Ejemplo: **estructurado** / **no estructurado**



2. **Paradigmas complementarios**
   - Pares de paradigmas con la particularidad de que no cumplir uno implica cumplir el otro
   - Ejemplo: paradigma **imperativo** y paradigma **declarativo**



Super-conjuntos: según clasificación (tipo 1)

3. **Paradigmas específicos**
   - Establecen una partición que afecta a todos los lenguajes
     - Cada lenguaje, cumple o no el paradigma
     - **No cumplir** el paradigma **no tendrá relevancia** para la caracterización de un lenguaje
   - Ejemplo: programación en **tiempo real**

- Clasificaciones de lenguajes solapadas

Super-conjuntos: según clasificación (tipo 2)

- La clasificación también puede ser según el tipo de restricciones que el paradigma establece en los lenguajes que lo cumplen. Dos categorías:

1. *Paradigmas orientados a establecer la misión del lenguaje o* "**Paradigmas de Objetivo**"

2. *Paradigmas orientados a establecer la forma que deberán tener los programas de un lenguaje o* "**Paradigmas de Estructura**"

Super-conjuntos: según clasificación (tipo 2)

- Paradigmas de Objetivo
  - Establecen el significado de las sentencias (contenido del código)
    - En su traducción a código máquina
    - Y en su representación de conceptos modelados en la fase de diseño
  - Un **ejemplo** de este tipo de paradigmas:
    - P. de **tiempo real**: describe capacidades de lenguajes que están relacionadas con conceptos de diseño y dificultades de implementación de programas
  - Más ejemplos:
    - Programación Concurrente
    - Programación Distribuida
    - Programación Imperativa

Super-conjuntos: según clasificación (tipo 2)

- Paradigmas de Estructura
  - Establecen recomendaciones de estructura de programas para evitar problemas inherentes u obtener ventajas (organización del código)
  - Ejemplos de estos paradigmas:
    - Programación Estructurada
    - Programación Modular
    - Programación Orientada a Módulos
    - Programación Orientada a Objetos
    - Programación Orientada a Procedimientos
    - Programación Orientada a Aspectos

Super-conjuntos: según la evolución

- La evolución de los paradigmas es la evolución de la programación
  - Primera línea evolutiva, el control de la memoria:
    - Imperativo -> Declarativo
  - Segunda línea evolutiva, la estructura del código:
    - Sin organización -> Estructurado -> Orientado a Objetos -> Orientado a Aspectos
  - Tercera línea evolutiva, número de tareas a la vez:
    - Secuencial -> Concurrente -> Distribuida -> Cliente/Servidor

Super-conjuntos: según relación con el HW

- Los lenguajes de programación son siempre abstracciones del HW
  - Podemos clasificar los paradigmas en función de su distancia al HW
  - Ejemplos:
    - Imperativo VS declarativo
      - Imperativo es como el HW actual
      - Declarativo, es más cercano al estilo humano, e independiente del HW actual
    - Concurrente
      - Más cercano o lejano en función de si el HW es paralelo o no paralelo
    - Orientado a Objetos
      - Lejano del HW

Algunos paradigmas

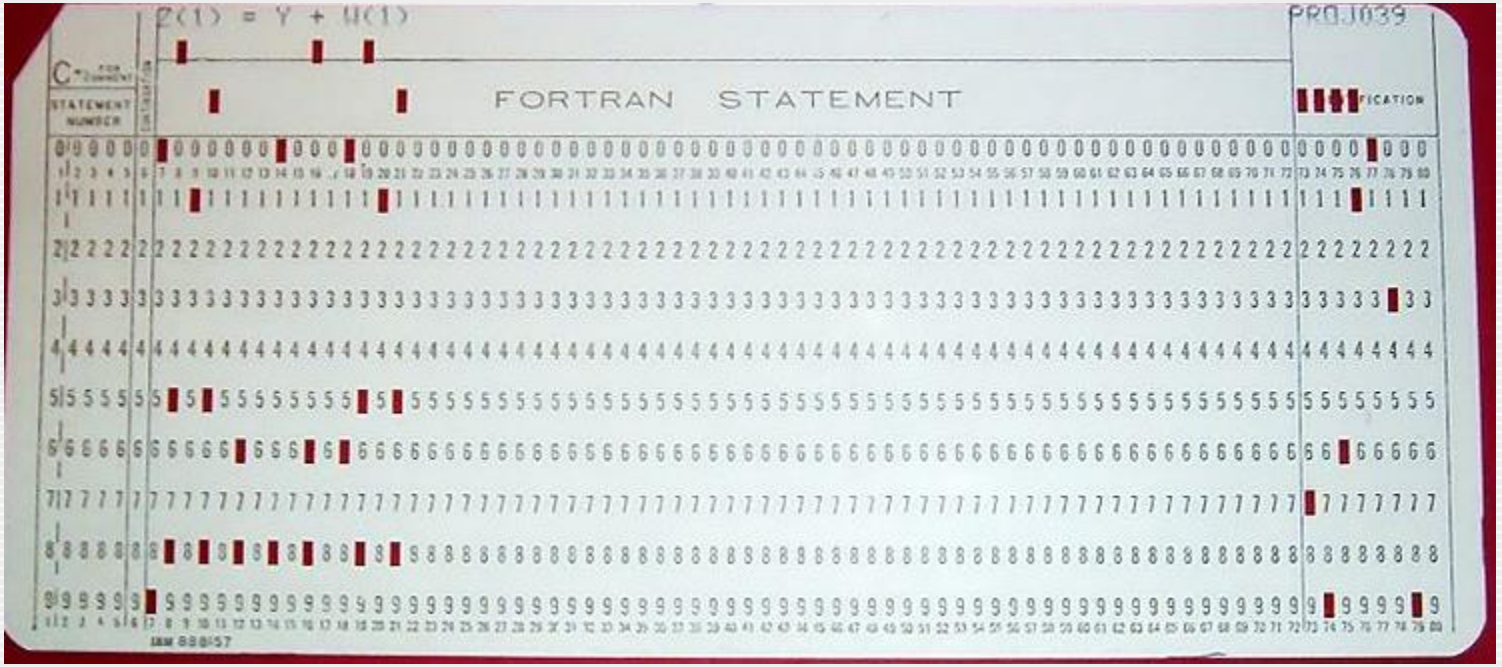
- Veremos los más importantes:
  - Programación imperativa
  - Programación declarativa
  - Programación concurrente
  - Programación orientada a objetos
  - Programación de tiempo real

Algunos paradigmas: imperativo

- Lenguajes **orientados a la acción** (máquina de Von Neumann)
- Describen "**cómo**" usar la memoria para obtener los resultados
  - Sentencias ejecutadas según un **control de flujo explícito**, que modifican el **estado** del programa
- Sentencias de:
  - **Asignación**: intervienen variables, expresiones, valores literales, funciones predefinidas, operadores
  - **Control de flujo**: saltos, bifurcaciones, iteraciones, invocaciones a subrutinas, tratamiento de excepciones

Algunos paradigmas: imperativo

- Ejemplos:
  - Fortran (1957), Algol (1960), Pascal (1971), C (1972), C++ (1985), o Java (1995)



Algunos paradigmas: declarativo

- Los programas describen "**qué**" se desea realizar con la memoria, pero no el manejo directo sobre ella
  - **No** hay control de flujo explícito

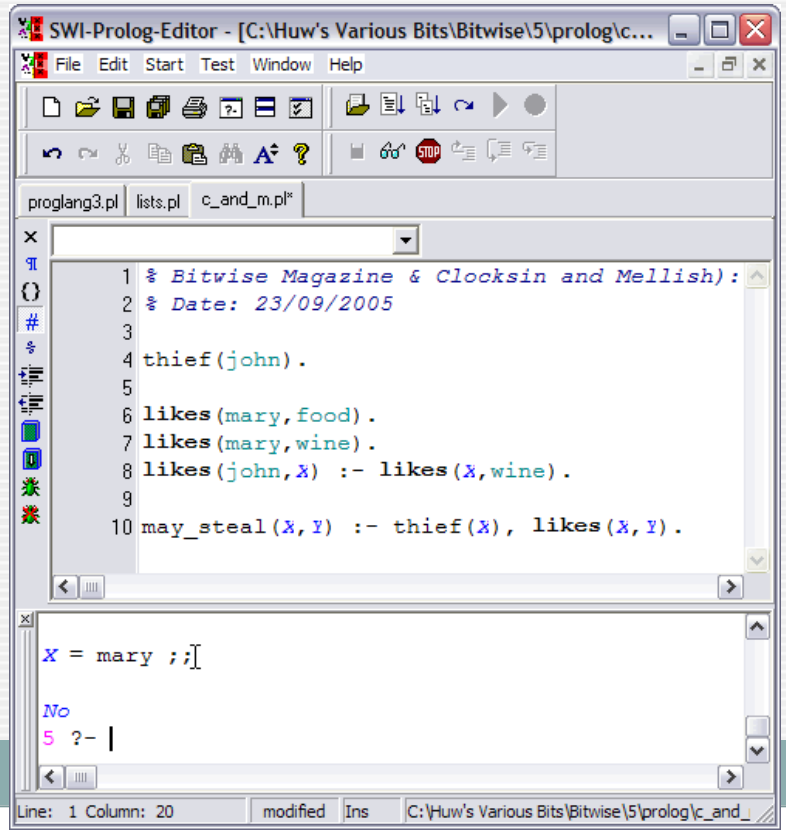
- Se puede dividir en dos:
  - **Funcional**:
    - Se modelan mediante funciones y expresiones.
    - No hay variables. No hay procedimientos explícitos.
    - Uso extensivo de la recursión.
  - **Lógico**:
    - Hechos, predicados y relaciones como medio para representar las tareas que se quieren realizar.
    - Basado en la lógica de predicados.

Algunos paradigmas: funcional

- Ventajas:
  - Gran expresividad
  - Extensibilidad
  - Facilidad de corrección
  - Ausencia de efectos laterales
- Inconvenientes:
  - Ineficiente (primeras implementaciones)
  - Complicada (recursividad)
- Ejemplos:
  - LISP (1959), APL (1962), ISWIM (1966), SCHEME (1975), FP (1977), HOPE (1980), CAML(1985), MIRANDA (1985), ML (1986), HASKELL (1988), Scala (2004)

Algunos paradigmas: lógico

- Basada en el razonamiento lógico
- Sistemas que tratan de responder a las demandas del usuario
- Manejan relaciones entre datos (hechos) en vez de funciones
- Ejemplo:
  - PROLOG (1972)



Algunos paradigmas: concurrente

- Capaces de realizar **simultáneamente** varias tareas
- Algoritmos que permitan dividir el trabajo en tareas
- Uso de técnicas, lenguajes, librerías y hardware específico, para conseguir el paralelismo
- Dos enfoques distintos de paralelismo:
  - Programación **concurrente** de memoria compartida
  - Programación concurrente **distribuida**

Algunos paradigmas: conc. memoria común

- Múltiples tareas de ejecución simultánea sobre una única máquina con **una única memoria**
  - El comportamiento es independiente del número de procesadores
  - El rendimiento aumenta con el número de procesadores
  - La concurrencia plantea problemas nuevos

Algunos paradigmas: conc. distribuido

- Se supera la barrera de la máquina única y la memoria única
- Los procesos intercambian información a través de una **red de comunicaciones**
- El paradigma más utilizado es el que implementa el modelo **Cliente-Servidor**

Algunos paradigmas: orientado a objetos

- Evolución del Paradigma Estructurado
  - El **Objeto** como entidad central del paradigma
  - **Objeto** = tipo de datos que engloba características (**estructuras de datos**) del objeto del mundo real
  - Los objetos modelan las características (**atributos**) y los algoritmos para manipularlas (**métodos**)

Algunos paradigmas: orientado a objetos

- Ventajas:
  - **Encapsulación** => alto grado de **reutilización** del código (se incrementa con la herencia)
  - **Representación** más directa del mundo real en el código (**abstracción**)
  - Se adapta mejor a la informática distribuida y a los modelos cliente/servidor
  - Se pueden crear **desarrollos más flexibles** y el proceso es más rápido
- Ejemplos:
  - Simula (1967), Smalltalk (1980), Eiffel (1988), muchos de los actuales (C++, Objetive C, CLOS, Ada o Java)

Algunos paradigmas: de tiempo real

- Proporciona las técnicas para controlar el funcionamiento de sistemas bajo ciertas **restricciones de tiempo**
- Características:
  - **Control** permanente y simultáneo sobre la ejecución
  - Manejo de **interrupciones** para cumplir las restricciones
  - Necesitan **acceder** directamente **al HW** de E/S
- La mayoría son aplicaciones de **control**
  - El sistema reacciona ante un cambio de estado

Ejemplos de programas

- Modula-2 (1978)

```
(* Copyright (C) 1987, 1990 Jensen & Partners
International *)

IMPLEMENTATION MODULE Strings;

PROCEDURE Assign(source:ARRAY OF CHAR;
                 VAR dest:ARRAY OF CHAR);
BEGIN
  Str.Copy(dest,source);
END Assign;

PROCEDURE Insert(substr:ARRAY OF CHAR;
                 VAR str:ARRAY OF CHAR;
                 inx:CARDINAL);
BEGIN
  Str.Insert(str,substr,inx);
END Insert;

PROCEDURE Pos(substr : ARRAY OF CHAR;
              VAR str : ARRAY OF CHAR):CARDINAL;
BEGIN
  RETURN Str.Pos(str,substr);
END Pos;
```

```
PROCEDURE Copy(str : ARRAY OF CHAR ;
               inx : CARDINAL ;
               len : CARDINAL ;
               VAR result : ARRAY OF CHAR);
BEGIN
  Str.Slice(result,str,inx,len);
END Copy;

PROCEDURE Concat(s1,s2 : ARRAY OF CHAR;
                 VAR result : ARRAY OF CHAR);
BEGIN
  Str.Concat (result,s1,s2);
END Concat;

END Strings.
```

Ejemplos de programas

- Python (1991)

```
>>> # Measure some strings:
... a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print x, len(x)
...
cat 3
window 6
defenestrate 12
>>>
--------------
>>> def f(x): return x%2 != 0 and x%3 != 0
...
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
>>>
```

Ejemplos de programas

- Visual Basic (1991)

```
VERSION 2.00
Begin Form FontDialog
   AutoRedraw   =   -1  'True
   Caption      =   "Select Font"
   ClipControls =   0   'False
...
End

Option Explicit
Dim Shared FntNum As Integer

Sub cmdCancel_Click ()
    ' Hide dialog
    FontDialog.Hide
End Sub

...
```

```
Sub lstMatchFonts_Click ()
    FntNum = lstMatchFonts.ListIndex
    lblFontDemo.FontName =
lstMatchFonts.List(FntNum)
End Sub

Sub lstMatchFonts_DblClick ()
    FntNum = lstMatchFonts.ListIndex
    lblFontDemo.FontName =
lstMatchFonts.List(FntNum)
End Sub
```

Ejercicios

- 1.- Establecer el nivel de cumplimiento del paradigma estructurado de los siguientes lenguajes de programación:
  - Pascal Java Fortran C Modula-2
  - Basic C++ Ada Caml
- 2.- Establecer los paradigmas que cumplen y el nivel de cumplimiento para los siguientes lenguajes de programación (consultando bibliografía o realizando búsquedas en Internet):
  - Pascal Java Fortran C Modula-2
  - Basic C++ Ada Caml Cobol
  - Lisp Prolog Parlog OCaml SmallTalk
  - Algol Ensamblador CLOS Python

Ejercicios

- 3.- Localizar la mayor cantidad posible de lenguajes de programación, distintos de los anteriores, y establecer sus características mediante el cumplimiento de paradigmas de cada uno de ellos. (De nuevo, se utilizará la bibliografía y la consulta a través de Internet).

- 4.- Escribir un programa que permita determinar si un número entero es primo o no, utilizando al menos cuatro lenguajes de programación diferentes.

  Nota: El número entero a examinar puede pasarse como parámetro, pedirse por el programa, ser el valor de una variable, tomarse de un cuadro de texto, etc.