



PROGRAMACIÓN II – UNIDAD 1

Ing. Gastón Weingand (gaston.weingand@uai.edu.ar)

Agenda

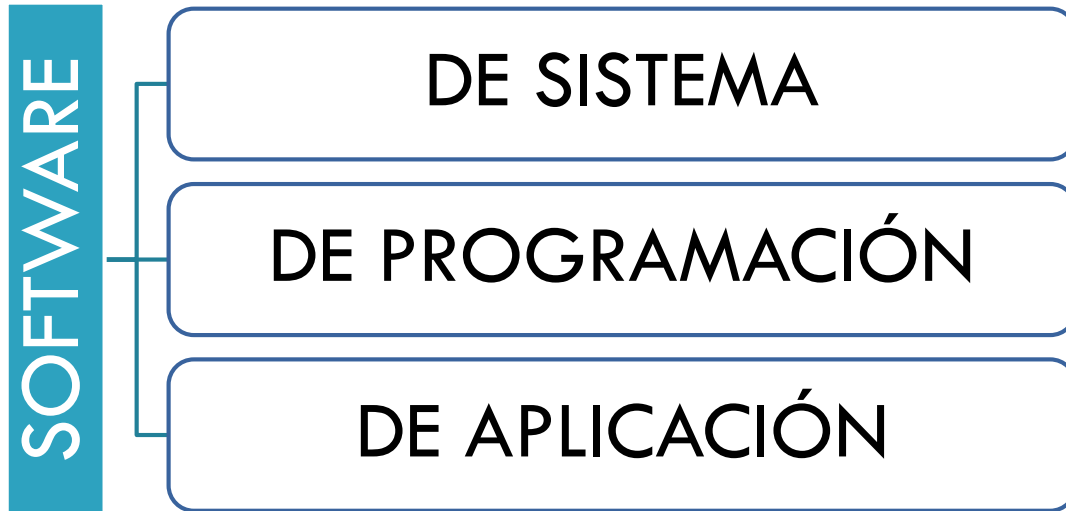
1. El Software.
2. Entorno integrado de desarrollo. Programas Vs. Proyectos
3. Aspectos distintivos de la programación orientada a objetos Vs. La programación estructurada. Vs. Programación funcional. Ventajas y desventajas
4. Abstracción de datos. Objetos y Clases. Herencia. Herencia múltiple. Polimorfismo. Lenguajes orientados a objetos/funcionales.
5. El lenguaje Python. Historia y evolución del lenguaje.

Agenda

1. **El Software.**
2. Entorno integrado de desarrollo. Programas Vs. Proyectos
3. Aspectos distintivos de la programación orientada a objetos Vs. La programación estructurada. Vs. Programación funcional. Ventajas y desventajas
4. Abstracción de datos. Objetos y Clases. Herencia. Herencia múltiple. Polimorfismo. Lenguajes orientados a objetos/funcionales.
5. El lenguaje Python. Historia y evolución del lenguaje.

1. El software

RAE: El software es un **conjunto de programas, instrucciones y reglas informáticas** que permiten ejecutar distintas tareas en una computadora.



1. El software



Pressman: El software distribuye el producto más importante de nuestro tiempo: LA INFORMACIÓN.

Industria del software -> Factor dominante en las economías del mundo industrializado

1. El software

Construcción de software -> Ing. Del Software

- ¿Por qué se requiere tanto tiempo para terminar el software?
- ¿Por qué son tan altos los costos de desarrollo?
- ¿Por qué no podemos detectar todos los errores antes de entregar el software a nuestros clientes?
- ¿Por qué dedicamos tanto tiempo y esfuerzo a mantener los programas existentes?
- ¿Por qué seguimos con dificultades para medir el avance mientras se desarrolla y mantiene el software?

Agenda

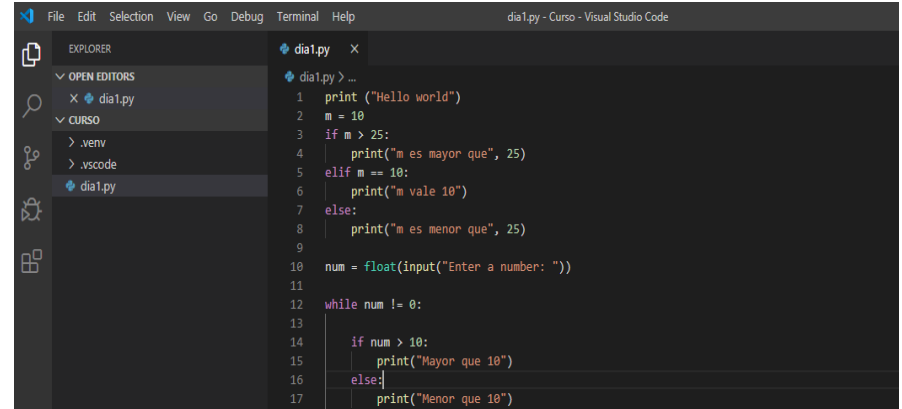
1. El Software.
2. Entorno integrado de desarrollo. Programas Vs. Proyectos
3. Aspectos distintivos de la programación orientada a objetos Vs. La programación estructurada. Vs. Programación funcional. Ventajas y desventajas
4. Abstracción de datos. Objetos y Clases. Herencia. Herencia múltiple. Polimorfismo. Lenguajes orientados a objetos/funcionales.
5. El lenguaje Python. Historia y evolución del lenguaje.

2. IDE. Programas vs. Proyectos

- *Integrated Development Environment (IDE)* o *entorno integrado de desarrollo* es un software que proporciona servicios integrales para facilitar el desarrollo de software a los programadores.
- Generalmente consiste en:
 - Editor de código fuente
 - Herramientas para la construcción automática
 - Depurador
 - IntelliSense
 - Compilador/Interprete

2. IDE. Programas vs. Proyectos

- *VISUAL STUDIO*
- *VISUAL STUDIO CODE*
- *IDLE*
- *Pycharm*
- *PyDev*
- <https://trinket.io/python/>
- <https://python.microbit.org/>



```
File Edit Selection View Go Debug Terminal Help
dial.py - Curso - Visual Studio Code

EXPLORER
  OPEN EDITORS
    X dial.py
  CURSOR
    .venv
    .vscode
    dial.py

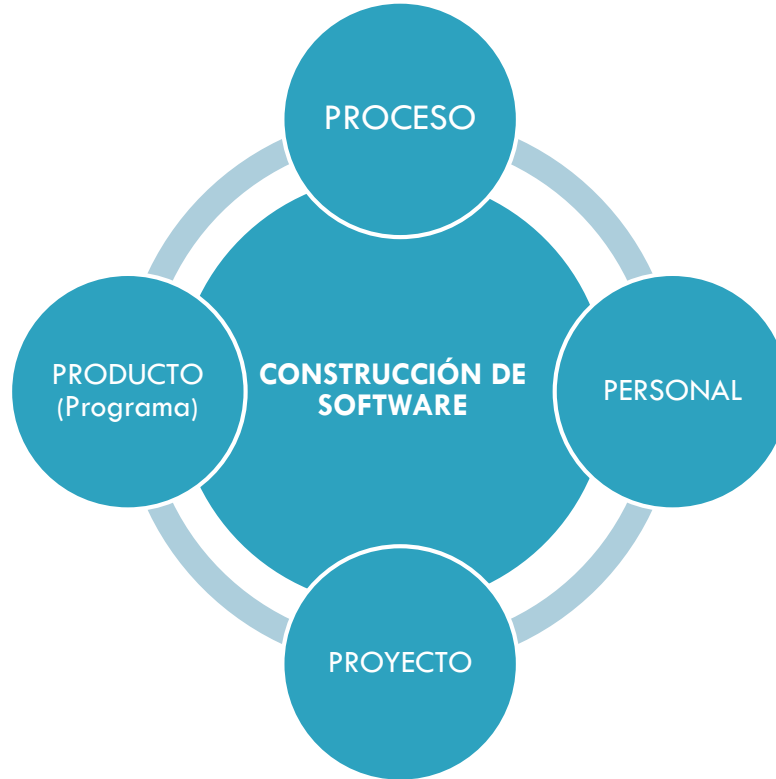
dial.py
1 print ("Hello world")
2 m = 10
3 if m > 25:
4     print("m es mayor que", 25)
5 elif m == 10:
6     print("m vale 10")
7 else:
8     print("m es menor que", 25)
9
10 num = float(input("Enter a number: "))
11
12 while num != 0:
13     if num > 10:
14         print("Mayon que 10")
15     else:
16         print("Menon que 10")
17
```

2. IDE. Programas vs. Proyectos

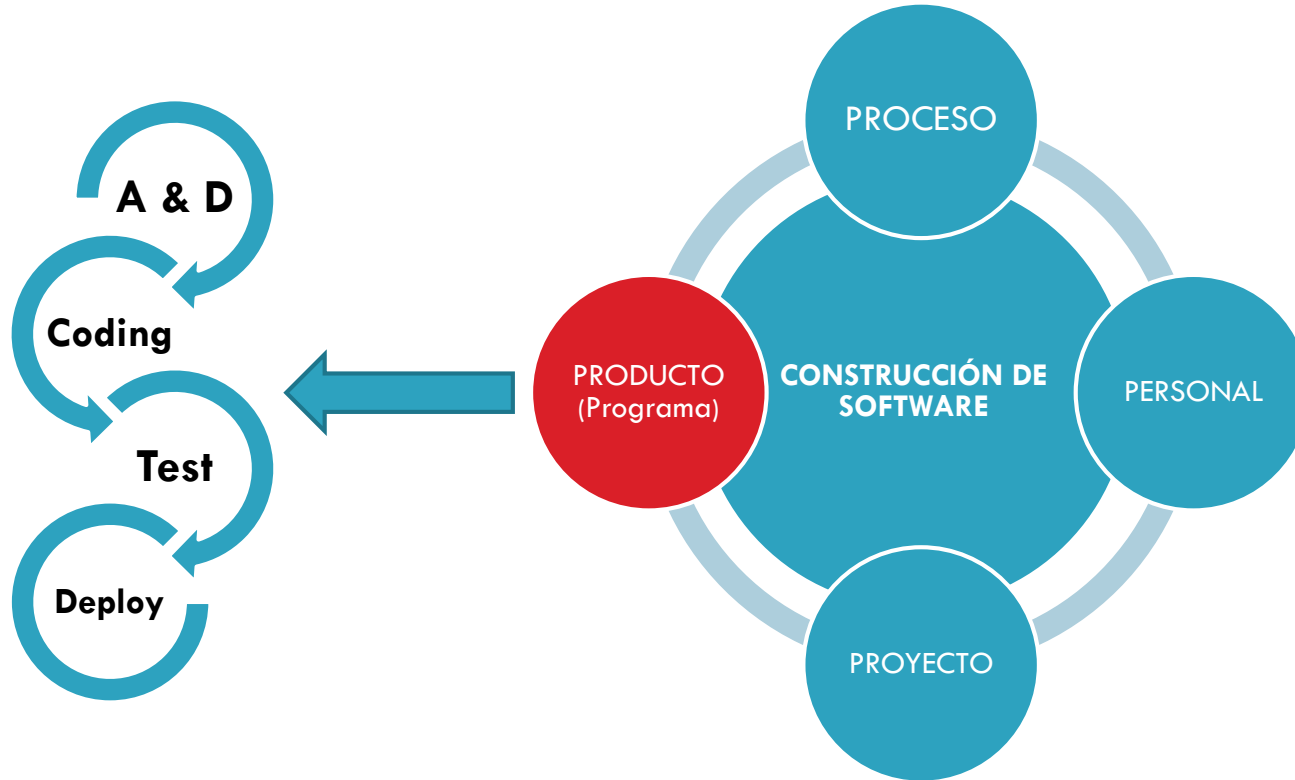
Configurar python con Visual Studio Code

1. [Visual Studio Code](#)
2. [Marketplace python](#) -> Descarga y mini tutorial de config.
3. Video explicativo inicial y entornos virtuales:
 - [Python en VS Code](#)

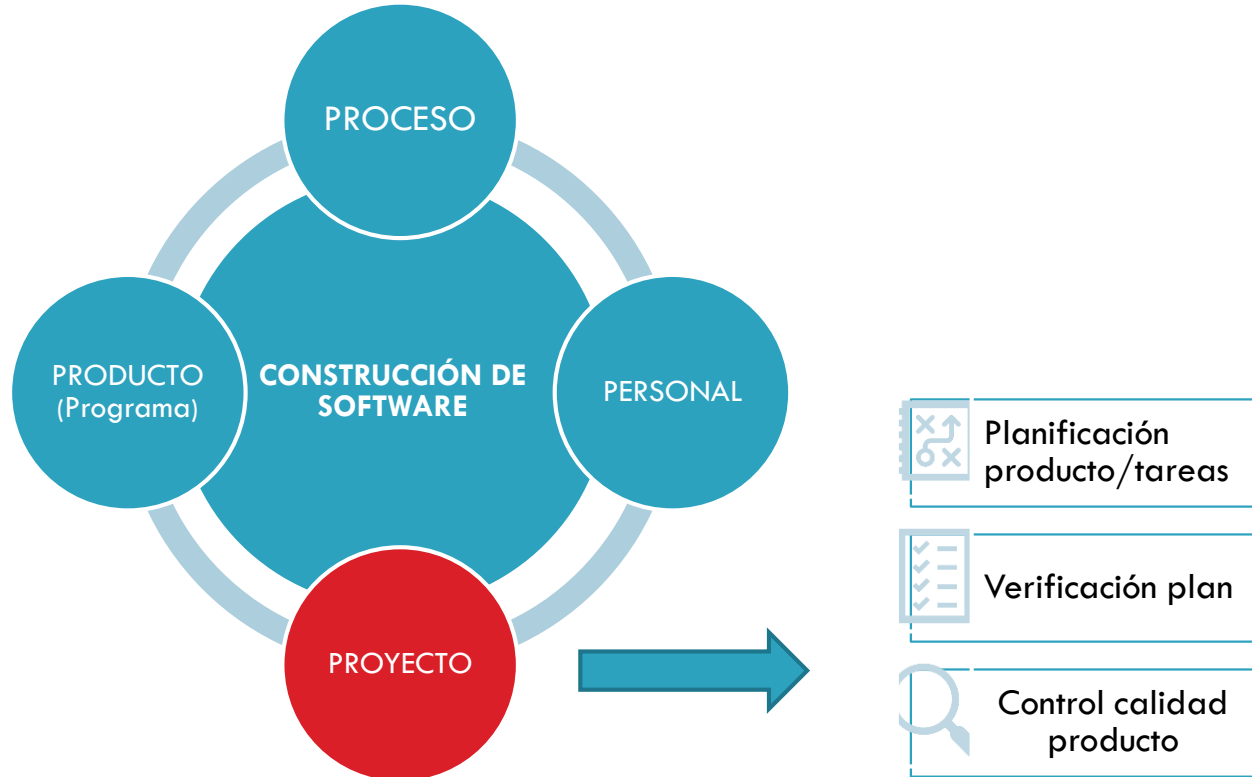
2. IDE. Programas vs. Proyectos



2. IDE. Programas vs. Proyectos



2. IDE. Programas vs. Proyectos



Agenda

1. El Software.
2. Entorno integrado de desarrollo. Programas Vs. Proyectos
3. Aspectos distintivos de la programación orientada a objetos Vs. La programación estructurada. Vs. Programación funcional. Ventajas y desventajas
4. Abstracción de datos. Objetos y Clases. Herencia. Herencia múltiple. Polimorfismo. Lenguajes orientados a objetos/funcionales.
5. El lenguaje Python. Historia y evolución del lenguaje.



Paradigmas

3-Paradigmas

Es un modelo, patrón o ejemplo.

“Teoría cuyo núcleo se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento”

Por ejemplo: Las leyes del movimiento o la teoría de la evolución.



¿Qué es un paradigma de programación?

Propuesta tecnológica adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados.

3-Paradigmas

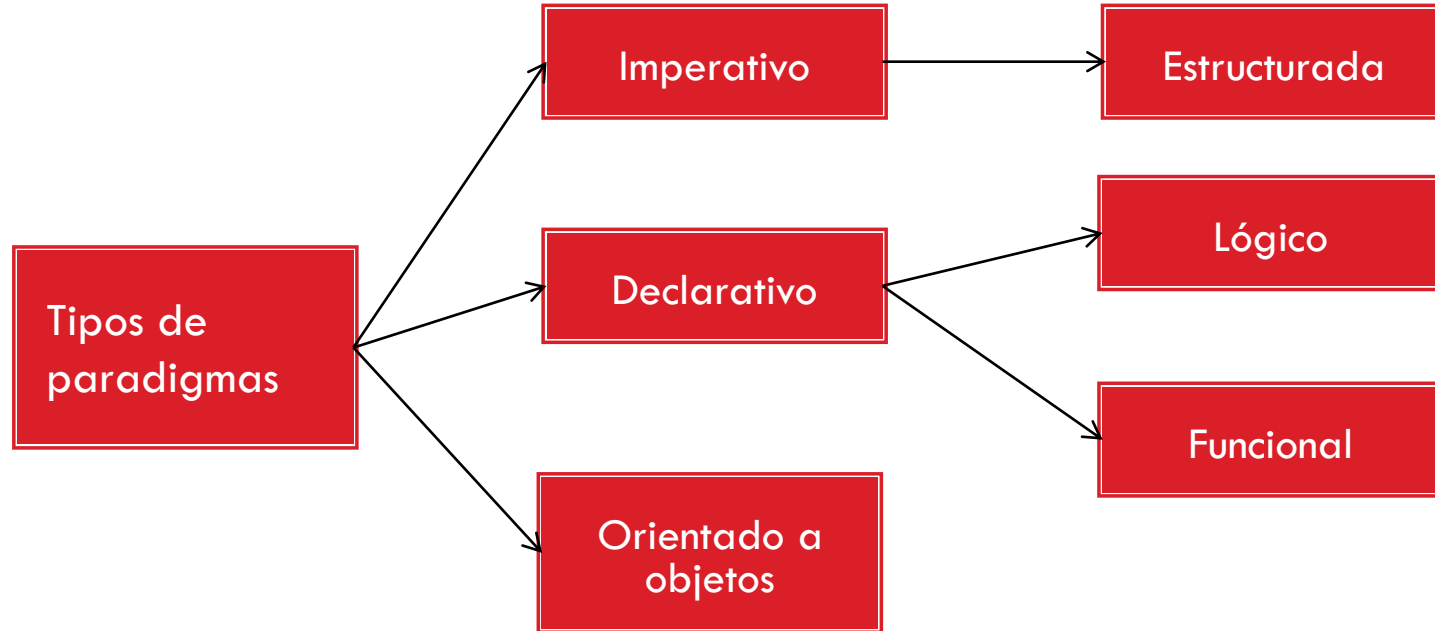
Un paradigma de programación define:

- Las herramientas conceptuales que se pueden utilizar para construir un programa (objetos, relaciones, funciones, instrucciones)
 - Las formas válidas de combinarlas
- Conjunto de ideas que modelan una solución a un problema dado.
-Mayor relación con el modelo mental que con los programas resultantes.

3-Paradigmas

- Los lenguajes de programación proveen implementaciones para los paradigmas.
- Existen lenguajes que se concentran en las ideas de un único paradigma, así como hay otros que permiten la combinación de ideas provenientes de distintos paradigmas.
- No es suficiente saber en qué lenguaje está construido un programa para saber qué marco conceptual se utilizó en el momento de construirlo.

3-Paradigmas



3-Paradigmas: Imperativo

- Especifica un programa en términos del **estado de programa y sentencias que cambian dicho estado**.
- Secuencia ordenada de instrucciones que manipulan un espacio de memoria.
- Es decir, este conjunto de instrucciones que le indican al computador cómo realizar una tarea. Lenguajes como Cobol, Pascal, C.
- Basado en **cómo se hace algo**.

3-Paradigmas: Imperativo

- Programa compuesto por un conjunto de instrucciones y variables que describen el estado del programa.
- Las instrucciones cambian el estado del programa.
- Un ejemplo de la vida real que se asemeje podría ser una receta de cocina.

3-Paradigmas: Declarativo



Está basada en la construcción de programas especificando un conjunto de condiciones, proposiciones, afirmaciones, que describen el problema.

La solución es obtenida mediante mecanismos internos de control.
Las variables son utilizadas con **transparencia referencial**.

Basado en **cómo es algo**.

3-Paradigmas: Declarativo

- A diferencia de la programación imperativa, donde se tiene un **algoritmo** en el que se describen los pasos necesarios para solucionar el problema, en la programación declarativa las sentencias describen el problema que se quiere solucionar, pero no las instrucciones para solucionarlo.
- La solución se encuentra mediante mecanismos internos de inferencia de información a partir de la descripción realizada.
- Ejemplos: HASKELL, PROLOG, SQL, HTML, XML

3-Paradigmas: Declarativo-> Funcional

Conjunto de funciones (Relaciones que cumplen las propiedades de unicidad y existencia), que pueden ser evaluadas para obtener un resultado

3-Paradigmas: Declarativo-> Lógico



Conjunto de predicados definidos a través de cláusulas (hechos y reglas) que describen propiedades y relaciones de un conjunto de individuos, sobre los cuales podemos realizar consultas.

3-Paradigmas: Orientado a objetos

Conjunto de objetos que se conocen entre sí a través de referencias y se envían mensajes en un ambiente

Ejemplos de lenguajes: C++, objective C, smalltalk, eiffel, ruby, python, C#, java, kotlin, vb.net.



CONCEPTOS TRANSVERSALES

3-Paradigmas: Abstracción

Una abstracción es una forma de interpretar y conceptualizar lo que resulta más importante de una entidad compleja, sin tener que tener en cuenta todos sus detalles.

Permite quedarse con lo esencial descartando lo que (A mi criterio, en ese momento) es accesorio.

En la vida cotidiana usamos abstracciones todo el tiempo, por ejemplo, podemos saber que una mesa es una mesa más allá de si es cuadrada o redonda, de madera o de plástico, con 4, 3 o 6 patas.

Cuando programamos es importante encontrar abstracciones.

3-Paradigmas: Abstracción

- Cada uno de los paradigmas nos va a brindar sus propias formas de abstracción.
- En programación estructurada, la forma de abstracción es el procedimiento. Un procedimiento permite tomar un conjunto de instrucciones y darles un nombre para poder utilizarla en otro contexto. Quien invoca el procedimiento se concentra en qué es lo que necesita resolver, el procedimiento es el que implementa el cómo se resuelve un determinado problema.
- La declaratividad es otra forma de abstracción, que nos permite describir el conocimiento relativo a un problema desentendiéndonos de los algoritmos necesarios para manipular esa lógica, que son provistos por el motor.

3-Paradigmas: Abstracción Alto Nivel vs Bajo Nivel

Cuando se habla de lenguajes de alto y bajo nivel, se refiere al nivel de abstracción de cada lenguaje.

Esta clasificación sirve para comparar entre diferentes lenguajes respecto a qué tan cercanas son sus abstracciones a la máquina (bajo nivel) o al programador (alto nivel).

Por ejemplo, Assembler trabaja directamente con instrucciones de máquina y los registros de la computadora, entonces podemos considerarlo como un lenguaje de más bajo nivel que C que es un lenguaje que se compila a código de máquina para permitirnos trabajar con procedimientos y estructuras complejas.

3-Paradigmas: Declaratividad

La declaratividad es una característica de algunas herramientas que permiten o fuerzan la separación entre el conocimiento del dominio de un programa y la manipulación de dicho conocimiento.

Dichas herramientas pueden ser de diversa naturaleza, tanto prácticas como lenguajes, entornos, frameworks, etc. o conceptuales como paradigmas o arquitecturas.

El concepto de declaratividad se contrapone a la tradicional programación imperativa en la cual el conocimiento y la lógica se encuentran muchas veces mezclados dentro de la misma porción de código resultando difícil determinar donde comienza uno y dónde termina el otro.

3-Paradigmas: Declaratividad

En un programa construido de forma declarativa se produce una separación entre la descripción del problema por un lado y los algoritmos o estrategias para encontrar la solución por el otro.

Ejemplo: Sistema de correlatividades de la facultad.

3-Paradigmas: Declaratividad

Pensando imperativamente, el algoritmo de solución sería algo así:

1. A partir de un alumno obtener la carrera que está cursando y con eso las materias de esa carrera, almacenarlas en una variable auxiliar.
2. Eliminar de esa colección de materias aquellas que el alumno ya haya cursado.
3. Recorrer la lista de las materias restantes y para cada una:
 - Obtener su lista de correlativas
 - Recorrerla y para cada correlativa verificar si el alumno cursó esa materia
 - En caso de no haberla cursado, eliminar la materia de la colección auxiliar

3-Paradigmas: Declaratividad

- En cambio, la definición declarativa eliminará todos los conceptos programáticos como variables auxiliares, recorrer colecciones o ir eliminando elementos de la colección.
- La versión declarativa no tendrá un concepto de orden, simplemente intentará describir el problema de la forma más abstracta posible, solamente tratando de contestar la pregunta, ¿qué materias puede cursar un alumno?
- Un alumno puede cursar las materias de su carrera que no haya cursado aún y cuyas correlativas sí haya cursado.

3-Paradigmas: Declaratividad

Un programa declarativo separa los siguientes elementos:

- El objetivo
- El conocimiento
- El motor que manipula el conocimiento para lograr el objetivo deseado

En el ejemplo anterior:

- Objetivo: obtener qué materias puede cursar un alumno.
- Conocimiento: El conocimiento es la información que se encuentra en la base de conocimiento sobre las materias disponibles en la carrera y cuáles ya cursó el alumno en cuestión.
- Motor: Toma el conocimiento y resuelve la consulta realizada en base al programa y deduce todas las posibles relaciones que la satisfagan.

3-Paradigmas: Declaratividad

- El lenguaje SQL para trabajar con bases de datos relacionales es un ejemplo de declaratividad que se usa en la industria.
- Los motores de bases de datos, a partir de consultas provistas por el usuario realizan búsquedas complejas relacionando las tablas de origen de la forma más optimizada posible.
- El algoritmo utilizado por el motor está muy separado del conocimiento (Qué entidades existen, cómo se relacionan entre ellas), lo cual permite al usuario abstraerse de esta lógica de búsqueda y concentrarse exclusivamente en el modelado de datos.

3-Paradigmas: Expresividad

- La expresividad puede definirse informalmente como “El nivel **estético** del código”.
- En otras palabras, escribir un código **expresivo** es poner atención a las cuestiones que hacen que este código fuente sea más fácil de *entender por una persona*.

3-Paradigmas: Expresividad

Comparemos estos dos códigos:

```
Function QuieroMoverElBote(a: Array of Integer, c: Integer): Real;  
Var b : Integer; d : Real; e : Integer;  
Begin  
  For b := 1 to c do  
    Begin  
      e := e + a[b];  
    End;  
  d := e / c;  
  QuieroMoverElBote := d;  
End.
```

```
Function Promedio(numeros: Array of Integer, cantidad: Integer): Real;  
Var i : Integer; sumatoria : Integer;  
Begin  
  For i := 1 to cantidad do  
    Begin  
      sumatoria := sumatoria + numeros[i];  
    End;  
  Promedio := sumatoria / cantidad;  
End.
```

- La computadora ejecutando este código produce exactamente el mismo resultado con cualquiera de los dos programas.
- La diferencia está en el programador que lee un programa o el otro. Se suele considerar a la expresividad como algo **subjetivo**.
- Pero hay formas de alcanzar la expresividad.

3-Paradigmas: Expresividad

Las técnicas que favorecen la *mejor comprensión* del código fuente por un programador, son técnicas que *no cambian en funcionamiento del programa*. Entonces, si el programa hace lo que corresponde, ¿por qué habríamos de consumir tiempo escribiendo código expresivo?

En la industria de software *los programas son cada vez más grandes, complejos, y cambiantes*.

En consecuencia, se necesita que:

- Sean flexibles, que puedan cambiarse fácilmente
- Que “fallen poco” (Con lo cual es importante encontrar y corregir errores tempranamente)
- El desarrollo dura mucho tiempo (meses, años)
- El equipo de desarrollo es amplio, mucha gente escribiendo el mismo programa.

3-Paradigmas: Expresividad

- Por lo tanto, un programador debe leer y corregir código existente (Propio o de otro) y en menor medida producir código nuevo.
- Es por esto que el código fuente *no puede ser exclusivamente escrito para la computadora*.
- El destino del código también son las propias personas.
- Es por eso que no se puede descuidar la expresividad: Es una de las varias formas de hacer la vida del programador más sencilla, para que pueda abordar la construcción de sistemas como el mencionado.

3-Paradigmas: Expresividad

¿Cómo lograr la expresividad?

- Usar **buenos nombres**: un nombre debe decir exactamente cuál es el propósito de la variable/ procedimiento/ método. Por ejemplo, la función anterior.
- Usar **buenas abstracciones** en general
- **Indentar** correctamente el código.
- Ser **descriptivo**: Por ejemplo, no tener miedo de escribir nombres largos. *cantAlumnosAprobados* es mejor que *aprobados*.
- Ser **claro y simple**: Por ejemplo, se pueden usar abreviaciones claras (Como *cant* en vez de *cantidad*). Aunque las abreviaciones pueden resultar a veces dañinas: *alumnosAprobados* es mejor que *alsAp*
- Respetar las **convenciones** es buena idea: Por ejemplo, el código escrito en lenguaje Python separa las palabras dentro de un nombre así: *esto_es_una_variable*, mientras que en Smalltalk la convención es así: *estoEsUnaVariable*.

3-Paradigmas: Orden Superior

Llamamos a una determinada operación de orden superior si la misma recibe otra operación por parámetro, siendo capaz de ejecutarla internamente.

Al usar orden superior tenemos las siguientes ventajas:

- Puedo aislar y **reutilizar comportamiento común**
- Puedo partir mi problema, **separando responsabilidades**, entre el código que tiene orden superior, y el comportamiento parametrizado
- Puedo tener un código con partes **incompletas**, esperando rellenarlos **pasando comportamiento por parámetro**, y no sólo datos.

3-Paradigmas: Orden Superior



Lo principal del orden superior es que “paso” algo más complejo que un número, una lista o una estructura (Functor, tupla, etc.) por parámetro.

Es decir, paso “comportamiento” por parámetro (predicado, función).

3-Paradigmas: Transparencia Referencial

$f(1)$ es una expresión E cuyo resultado da un valor V

Decimos que una solución tiene transparencia referencial si podemos reemplazar en un programa todas las expresiones E por el valor V sin alterar el resultado del programa, independientemente del lugar donde aparezca E .

No depende del contexto de ejecución, ni del orden de evaluación de dicha expresión.

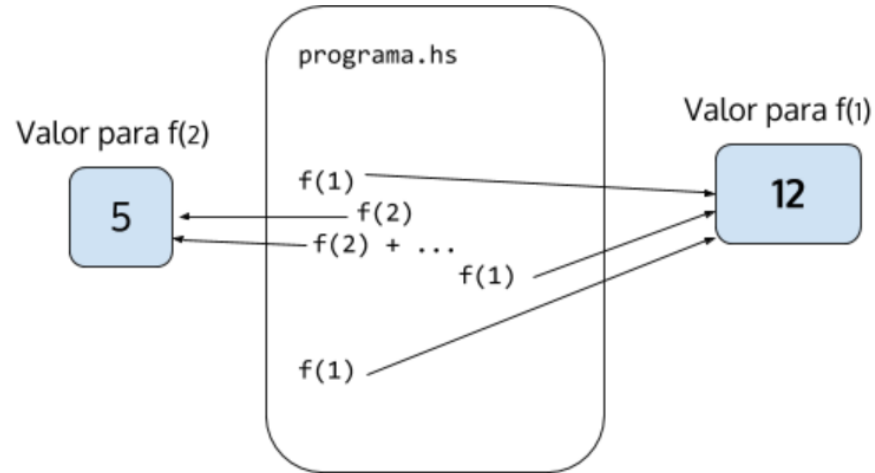
3-Paradigmas: Transparencia Referencial

Hay transparencia referencial cuando al realizar una operación con los mismos valores siempre da el mismo resultado.

Decimos que una operación tiene transparencia referencial si es:

- Independiente: No dependen del estado de nada que este fuera de sí misma
- Determinística: Siempre devuelven el mismo valor dados los mismos argumentos
- No produce efecto colateral

3-Paradigmas: Transparencia Referencial



3-Paradigmas: Veamos esto en código:

```
program Prueba
var
    flag: BOOLEAN

function f (n: INTEGER): INTEGER
begin
    flag ← NOT flag
    if flag then
        ↑ n
    else
        ↑ 2 * n
    end if
end
```

```
--- Programa Principal
begin
    flag <- true;
    ...
    write( f(1) );    <- escribe 2
    write( f(1) );    <- escribe 1
    ...
    write( f(1) + f(2) ); <- escribe 4
    write( f(2) + f(1) ); <- escribe 5
    ...
end
```

3-Paradigmas: Efecto Colateral

Hay efecto colateral cuando un cambio de estado se produce por la realización de una operación. Por ejemplo, una operación puede modificar una variable global, modificar uno de sus argumentos, escribir datos a la pantalla o a un archivo, o hacer uso de otras operaciones que tienen este efecto.

Otra definición válida es:

Si sacamos una foto al sistema (llamémosla F1), después se realiza la operación de su interés, y se le vuelve a sacar una foto al sistema (F2). Si F1 y F2 son distintas \Rightarrow la operación que se hizo tiene efecto colateral o de lado.

3-Paradigmas: Asignación Destructiva



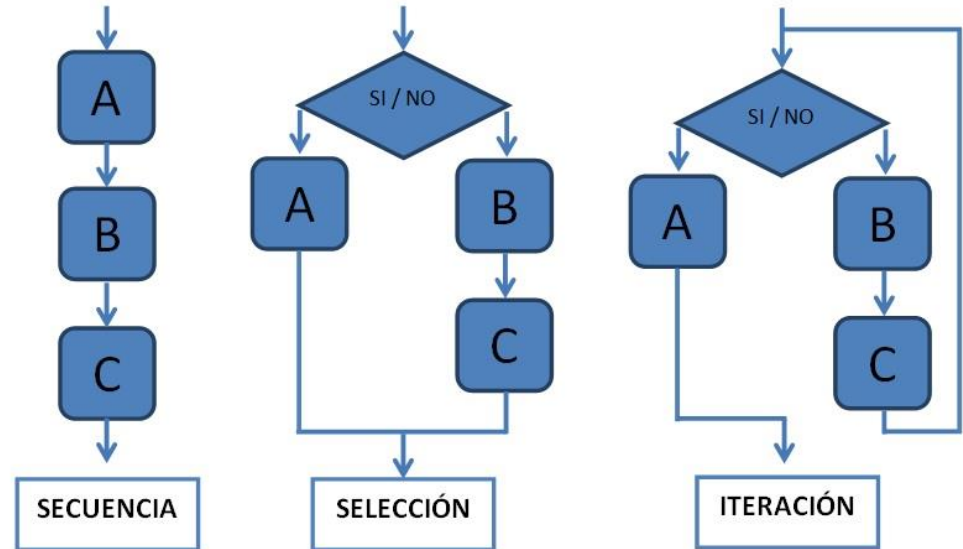
Asignar destructivamente es reemplazar el valor de una variable por otro valor.



PARADIGMA ESTRUCTURADO

3-Paradigmas: Paradigma estructurado

Teorema de [Corrado Böhm](#) y [Giuseppe Jacopini](#)



3-Paradigmas: Paradigma estructurado - Ventajas

- Menos esfuerzo en el diseño (Abstracciones)
- Menor esfuerzo en el testing
- La lectura (Entendimiento) de un programa suele ser más sencilla al ser del tipo secuencial
- Se obtiene mayor productividad en general

3-Paradigmas: Paradigma estructurado - Desventajas

- Monobloque de código: Complejidad de agregar nuevas funcionalidades sin producir daño colateral.
- Complejidad en la reutilización
- La cantidad de líneas de código suele ser mayor que en otros paradigmas



PARADIGMA ORIENTADO A OBJETOS

3-Paradigmas: Orientado a objetos

Es una técnica de programación que usa objetos y sus interacciones para diseñar aplicaciones.

Proporciona una forma de programar cercana a como expresamos las cosas en la vida real.

Se combinan los datos y los procedimientos en una entidad única.

3-Paradigmas: Orientado a objetos

Ver Presentación “Teoría de orientación a objetos”
para comprender los conceptos fundamentales

3-Paradigmas: Orientado a objetos - Ventajas

- Flexibilidad: A través de clases establecemos módulos independientes. Podemos pensar en estos módulos como bloques con los cuales podemos construir diferentes programas.
- Reuso: Por ejemplo, una vez que se define una entidad, esta se puede utilizar donde sea conveniente.
- Mantenibilidad: Las clases que conforman una aplicación, vistas como módulos independientes entre sí, son fáciles de mantener sin afectar a los demás componentes.
- Extensibilidad: Gracias a la modularidad permite extender para cubrir necesidades de crecimiento.

3-Paradigmas: Orientado a objetos - Desventaja

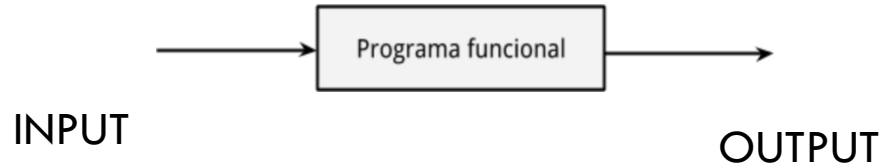
- En desarrollos muy simples a veces se requieren “pasos extra” para logra el objetivo
- Aumento de los requerimientos de memoria y procesos en general.
- Necesidad de documentación clara ya que las visiones de las abstracciones no suelen estar unificadas entre diferentes diseñadores



PARADIGMA FUNCIONAL

3-Paradigmas: Paradigma Funcional

Un programa según el Paradigma Funcional será una función. Es decir, una transformación de un INPUT en OUTPUT



3-Paradigma f.: Enfoques de función

Función como caja negra

Pensar una función es como una máquina con UNA salida y al menos una entrada, capaz de producir un resultado.

Decimos que se trata de una caja negra, porque para aquel que la use no tiene acceso al interior de la misma, sino tan solo a sus entradas y salida.

3-Paradigma f.: Enfoques de función

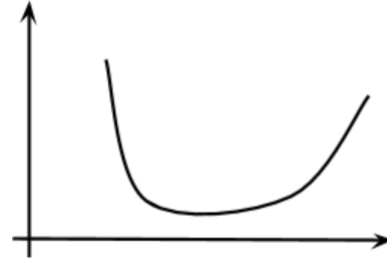
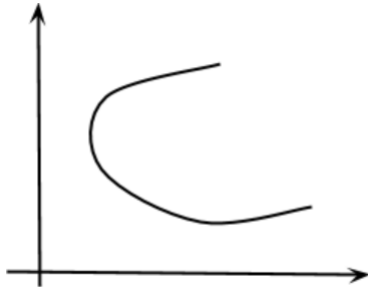
Función como transformación matemática

Las funciones son relaciones que presentan las siguientes características:

- para toda entrada aceptable (su dominio), existe un único resultado (imagen), lo cual se conoce como unicidad.
- para toda entrada del dominio, existe un resultado, lo que se conoce como existencia.

3-Paradigma Funcional

¿Cuál de estas dos imágenes es una función?



3-Paradigma f.: Concepto de Unicidad



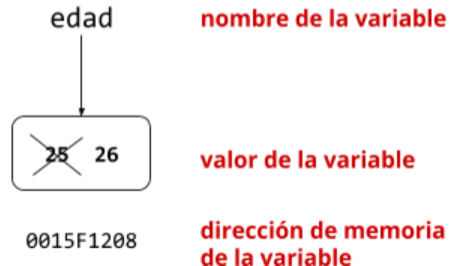
Implica que cada elemento de A está relacionado con solo un elemento de B .

Un elemento x de A no puede estar relacionado con dos elementos distintos de B .

3-Paradigma f.: Variable

En el paradigma imperativo una variable es una posición de memoria

`edad = edad + 1`



3-Paradigma f.: Variable

En el paradigma funcional, en cambio es un valor desconocido, o que todavía no fue calculado. Es la definición matemática de variable.

Por ejemplo:

$$f(x) = y = \log x$$

3-Paradigma f.: Ventajas

- Código más elegante: Compresión y expresividad claras
- La concurrencia es eficaz
- Bajo índice de errores
- Facilidad en la depuración y pruebas
- Evaluación diferida (El valor se evalúa y almacena solo cuando es necesario)

3-Paradigma f.: Desventajas

- Los valores inmutables y la recursividad suelen producir reducción en el rendimiento
- En algunos casos, escribir funciones puras provoca una reducción en la legibilidad del código
- Escribir programas en estilo recursivo en lugar de usar bucles para el mismo puede ser una tarea desalentadora

Agenda

1. El Software.
2. Entorno integrado de desarrollo. Programas Vs. Proyectos
3. Aspectos distintivos de la programación orientada a objetos Vs. La programación estructurada. Vs. Programación funcional. Ventajas y desventajas
4. Abstracción de datos. Objetos y Clases. Herencia. Herencia múltiple. Polimorfismo. Lenguajes orientados a objetos/funcionales.
5. El lenguaje Python. Historia y evolución del lenguaje.

5-Lenguaje python

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible.

Se trata de un **lenguaje interpretado** o de script, con **tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos.**

5-Lenguaje python

Python tiene muchas de las características de los lenguajes compilados, por lo que se podría decir que es **semi interpretado**. En Python, como en Java y muchos otros lenguajes, el código fuente se traduce a un pseudo código máquina intermedio llamado **bytecode** la primera vez que se ejecuta, generando archivos .pyc o .pyo (bytecode optimizado), que son los que se ejecutarán en sucesivas ocasiones.

5-Lenguaje python

¿Por qué python?

La sintaxis de Python es tan sencilla y cercana al lenguaje natural que Introducción los programas elaborados en Python parecen pseudocódigo. Por este motivo se trata además de uno de los mejores lenguajes para comenzar a programar

5-Lenguaje python

¿Por qué python?

La sintaxis simple, clara y sencilla; el tipado dinámico, el gestor de memoria, la gran cantidad de librerías disponibles y la potencia del lenguaje, entre otros, hacen que desarrollar una aplicación en Python sea sencillo y muy rápido

5-Lenguaje python

¿Por qué python?

Python no es adecuado sin embargo para la programación de bajo nivel o para aplicaciones en las que el rendimiento sea crítico.

Algunos casos de éxito en el uso de Python son Google, Yahoo, la NASA, Industrias Light & Magic, y todas las distribuciones Linux, en las que Python cada vez representa un tanto por ciento mayor de los programas disponibles.

5-Lenguaje python

Tipado dinámico

La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.

5-Lenguaje python

Fuertemente tipado

No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente.

5-Lenguaje python

Multiplataforma

El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios

5-Lenguaje python

Orientado a objetos

La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos. Python también permite la **programación imperativa, programación funcional y programación orientada a aspectos (POA)**.

5-Lenguaje python

Versiones (Dic-2019: 3.8.1)

