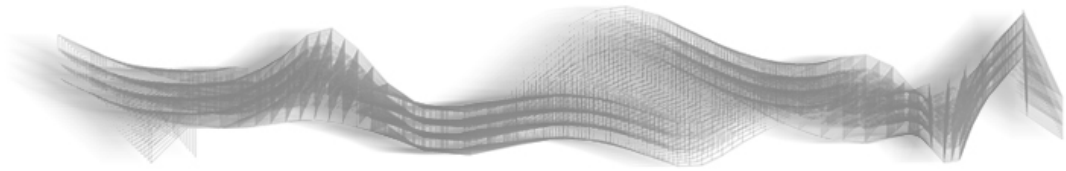
Soluciones de Diseño Computacional
Computational Design Solutions

Universidad de Chile :: Facultad de Arquitectura y Urbanismo :: Santiago de Chile ::
4 al 9 de Agosto 2007 :: www.espaciosdigitales.org/santiago/2



Instructores / *Instructors*

Kenfield Griffith
John Snavely
Daniel Cardoso (Workshop 2006)
Pablo C. Herrera (TA)

Descripción / *Description*

Soluciones de Diseño Computacional brinda a los estudiantes una forma de acercarse a la programación como una herramienta de exploración formal y espacial. El objetivo es dotar a los participantes de herramientas conceptuales y técnicas que les permitan decidir cuando y de que manera la programación puede volverse un aliado del proceso de diseño.
Los estudiantes exploraran las posibilidades de este lenguaje a través de una serie de ejercicios guiados y del desarrollo de un proyecto personal.

El Taller brindará bases sólidas de programación enfocándose en el lenguaje RhinoScript. Al final del Taller los estudiantes estarán en capacidad transmitir ideas espaciales generadas algorítmicamente.

The Computational Design Solutions Workshop is based on a view of scripting or programming as a tool to extend the horizon of available design possibilities. The goal is to provide students with the conceptual and technical tools to decide when one can apply scripting to a given architectural problem.
The students will be introduced to a set of strategies that designers can use in order to establish a productive dialogue with a computational medium in order to achieve particular design objectives.
The workshop will focus on RhinoScript language as the medium for this dialogue, and the students will explore it's possibilities through a series of guided exercises and the development of a personal project assisted by the instructors. At the end of the workshop the students will be able to visually convey computationally generated design ideas.

Introducción / *Introduction*

Los computadores son artefactos hechos por el hombre; por lo tanto imitan la manera en que las personas leen y piensan. El código es leído por la maquina de izquierda a derecha y de arriba a abajo, al igual que el Español, el Ingles. Cada línea de código es una instrucción que le dice al computador exactamente que *hacer*.

Computers are man-made devices, therefore they mimic the way humans read and think; code is read from left to right and from top to bottom (like Spanish, English, etc.). Each line of code it tells the computer to do something.

Ejemplo / *Example*

primera línea de código / *first line of code*
segunda Línea de código / *second line of code*
funcionA() / *some function()*
mas código / *continue code*
funcionB() / *some function()*
mas codigo
etc.

Recursos Disponibles / *Resources*

Página del Taller Agosto 2007
www.espaciosdigitales.org/santiago/2

VBScript Reference
http://msdn2.microsoft.com/en-us/library/sh9ywfdk(en-US,VS.80).aspx

Lista de métodos específicos de RhinoScript
http://www2.rhino3d.com/scripting/

Manual de RhinoScript por David Rutten
http://www.rhino3d.com/tutorials/vbscript/RhinoScript.pdf

Más métodos, documentados por Stelios Dritsas
http://www.dritsas.net/scripting/library/

Sintaxis / *Syntax*

Concepto / *Concept*

Como en cualquier lenguaje, los lenguajes de programación tienen su propia sintaxis. Sin ella, la comunicación entre el computador y el programador seria imposible, y ninguna instrucción de este podría ser entendida (compilada) y ejecutada por aquel.

As any language, programming languages have their own specific syntax rules. Without these rules the communication between the computer and the programmer would be impossible, and no instruction from the programmer could be understood (compiled) and executed by the machine.

Un lector humano esta acostumbrado a interpretar las ambigüedades del lenguaje, y podría entender sin mucha dificultad cualquiera de las frases a continuación, a pesar de que solo una de ellas es correcta de acuerdo a la sintaxis del idioma español.

Every programming language has its own set of syntax rules. A human reader, used to deal with ambiguity, could easily understand any of these sentences even though only the first one fully complies with the syntax rules of Spanish language:

Muchos de los presentes prefirieron levantarse de la mesa y salir	*Muchos de los presentes prefirieron levantarse de la mesa y salir*
De los presentes muchos levantarse de la mesa y salir prefirieron	*Muchos de los Presentes prefirieron Levantarse de la mesa y salir*
Muchosdelos presntes prfirieron levantarse de lamesaysalir	*Muchosdelos presntes prfirieron levantarse de lamesaysalir*

Por contraste, un computador necesita instrucciones que se ajusten de manera estricta a la sintaxis del idioma en el que fueron creadas. *RhinoScript* fue creado basado en un lenguaje mas amplio llamado *Visual Basic*, y por lo tanto la sintaxis de ambos es basicamente la misma.

A computer, in contrast, needs instructions that follow strictly and unambiguously the syntax rules of the language. Rhinoscript is built on a larger programming language called Visual Basic, and therefore its syntax rules are basically the same.

Ejemplo	***Example***
Para una referencia completa de Visual Basic http://msdn2.microsoft.com/en-us/library/sh9ywfdk(en-US,VS.80).aspx For a complete VB Script Syntax reference	*Refer to:* *http://msdn2.microsoft.com/en-us/library/sh9ywfdk(en-US,VS.80).aspx* *For a complete VB Script Syntax reference*

Variables / *Variables*

Concepto / *Concept*

La noción de variable está presente en todos los lenguajes de programación, y hace referencia a 'lugares de memoria' que guardan los valores que hemos de manipular en nuestros programas. Podemos imaginarlas como cajas que guardan objetos. Cuando escribimos un programa, creamos variables para darle un espacio en la memoria del computador a un determinado objeto o valor. A manera de ejemplo simple, las variables son útiles para acceder a valores que no están disponibles en el momento en que el programa es escrito, como por ejemplo valores que el usuario ingresa en el sistema, o los resultados de un cálculo interno del código. Un programa puede tener muchas variables, así que es muy importante que al nombrarlas se tenga en cuenta el tipo de información que la variable ha de guardar.

The concept of variable is present in every programming language. Variables are memory locations that store the values we need to manipulate in our computations. We can think of them as boxes or buckets that store data. We create variables in order to allocate space for an item inside the computer's memory. Variables are useful for accessing values that are not available at the moment the script is written, such as user input values, or results of internal calculations of the program. A script can have many variables; that's why it's very important that the names we give them clearly address the specific nature of the value or values we are storing in them.

Ejemplo de nombres de variables

```
'sumaTotal', 'promedio', 'arcosPrincipales'
```

Variable names Example

'totalSum', 'average', 'mainArches'

Ejemplos de uso algebraico de variables

2x + 1 = 5
sabemos que la variable x = 2

ejemplo: (calculadora)
2x + 1 = y
'x' y 'y' están representando instancias de un valor, de modo que 'y' esta siendo afectado por x.

Here we see an example of the algebraic use of variables

2x + 1 = 5
we know that variable x = 2

example: (calculator example)
2x + 1 = y
x is representing an instant, so is y
therefore you want the results of y to be influenced by x

Ejemplo de Sintáxis

- dim variableName 'crea espacio para un valor
- redim arrayName(numberOfElements) 'crea espacio para una colección de variables (array)

Syntax Example

- *dim variableName 'allocates space for a value*
- *redim arrayName(numberOfElements) ' allocate space for an array (a collection of values)*

Las variables pueden ser de distintas clases, como por ejemplo cadenas de caracteres *strings*, números enteros *integers*, números reales *doubles*, etc. En la sección siguiente veremos en detalle algunos de los tipos de variable más comunes.

Variables can be strings, integers, doubles, floats, etc. In the next section we will see some of the most commonly used data types.

Tipos de datos / *Data Types*

Un programa involucra normalmente distintos tipos de información. A veces necesitamos hacer cálculos con números enteros, otras veces necesitamos usar palabras, o establecer el valor de falsedad e una acción o afirmación [VERDADERO/FALSO].
Cada tipo de dato tiene implicaciones en la cantidad de memoria que es necesaria para su almacenamiento. Esto hace que la precisión y eficiencia de un programa esté relacionada con la asignación y el uso correcto de estos tipos de datos. Algunos de estos tipos están descritos a continuación.

Scripts typically involve more than one type of information; sometimes for our computations we need to manipulate integer numbers, words, or to store the assessment of an action [TRUE/FALSE]. Each data type has implications on the amount of memory that is used to allocate the value. The precision and efficiency of a computation is linked to the correct use of data types.
Some of the most commonly used data types are
1 bit = 0 [can toggle between 1 and 0]
1 byte = 8 bits

Tipo de dato :: Verdadero o Falso / *Boolean True or False*

Usa 1 bit de memoria
Rango: Verdadero o Falso
Ejemplo: T (true = verdadero, false = falso)

Uses 1 bit of memory
Range: True or False
Example: true

Tipo de dato :: Números Reales / *Double, Float*

Usa 8 bytes de memoria 2^6
[1010101010101010101010010101 0101010101010101010101010010101010]
Rango: -1.79769313486231570E+308 a -4.94065645841246544E-324 † para valores negativos, y
4.94065645841246544E-324 a 1.79769313486231570E+308 † para valores positivos.
Ejemplo: 3.14159265358979323846264338327

Uses 8 bytes of memory 2^6
[1010101010101010101010010101 0101010101010101010101010010101010]
Range: -1.79769313486231570E+308 through -4.94065645841246544E-324 † for negative values;
4.94065645841246544E-324 through 1.79769313486231570E+308 † for positive values
Example: 3.14159265358979323846264338327

Tipo de dato :: Número entero / *Integer*

Usa 4 bytes de memoria 2^5
Rango: -2,147,483,648 a 2,147,483,647
Ejemplo: 1,980,300

Uses 4 bytes of memory 2^5
[10101010101010101010101001010101]
Range: -2,147,483,648 through 2,147,483,647 (signed)
Example: 1,980,300

Tipo de dato :: Número entero / *Long*

Usa 8 bytes de memoria 2^6
Rango: -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 (9.2...E+18 †)
Ejemplo: 600,000,000,000,000,000

Uses 8 bytes of memory 2^6
[1010101010101010101010100101010101 0101010101010101010101010010101010]
Range: -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807 (9.2...E+18 †) (signed)
Example: 600,000,000,000,000,000

Cadena de caracteres / *String*

El uso de la memoria depende de la cantidad de caracteres de la cadena
Ejemplo: “Hola”

The use of memory depends on the implementation
Example: “Hola”

Estructuras de datos / *Data Structures*

Colecciones, o arreglos / *Array*

Los arreglos son los miembros principales de un conjunto de estructuras llamadas 'Listas'. Estas estructuras pueden alojar un gran número de objetos en lugar de solo uno. Podemos imaginar un arreglo como una caja de huevos; la caja es el arreglo, y los huevos los miembros del arreglo. Los arreglos pueden alojar un número ilimitado de objetos. Al este número se le conoce como el 'tamaño' del arreglo.

Arrays are important members of a larger set of data structures called *Lists*. These data structures can allocate many objects instead of only one. One can think of an array as a carton of eggs; the carton being the array (holder) and the eggs being the objects (members) of the array. An array can have any size.

Arreglos de diferentes tamaños / *Different sizes of arrays*

Arreglo de tamaño 12 / *Array of size 12*

Arreglo de tamaño 6 / *Array of size 6*

Otros tipos de listas son las listas vinculadas, y las Vlistas que almacenan los objetos de manera levemente distinta. Sin embargo, para efectos de este taller, vamos a enfocarnos en los arreglos.
Para qué son útiles los arreglos?

Other lists include linked lists and Vlists, that index the objects in slightly different ways. But for the purposes of the workshop we will focus on arrays.
What are arrays useful for?

- Para darle a los objetos una 'dirección' que nos permita referirnos a ellos de manera sistemática y eficiente
- Para efectuar una determinada operación en un número grande de objetos

- *For giving objects a known address that lets us refer to and manipulate an object in precise terms.*
- *For efficiently performing a certain operation to a large number of elements or values*

Ejemplo de declaración de un arreglo

Example

```
redim coleccionDeCoordenadas([número de elementos])
redim arrayName(numberOfElements)
```

Ejercicio Guiado # 1

- Declarar variables
- Declarar arreglos
- Asignar valores a las variables, imprimirlos
- Hacer operaciones simples en las variables, e imprimir los resultados
- Leer el contenido de una posición específica en un arreglo, e imprimirlo

Guided Exercise 1

- *Declaring Variables*
- *Creating Arrays*
- *Assigning values to variables, reading them and printing them*
- *Making simple computations with variables, and printing the results*
- *Reading the contents of a particular array position, and printing it*

Condicionales / *Conditionals*

Las instrucciones condicionales, como su nombre lo indica, se ejecutan solo cuando una determinada condición se cumple. Son una de las nociones más versátiles, fundamental en cualquier lenguaje de programación.

Conditional statements are key to programming. They are requests to the computer to execute an instruction if a certain given condition is met.

Sintaxis / *Sintax*

Ejemplo 1 / *Example 1*

```
If ( variable = TRUE )

      Ejecutar instrucciones

End If
-------------------------------------
```

En el ejemplo de arriba, evaluamos si una función se cumple por medio de una variable booleana (ver sección anterior, ejemplos de tipos de datos) que puede ser verdadera o falsa.

In the above example we check if a condition is met using a boolean variable (see above section on variables) that can be either TRUE or FALSE.

Ejemplo 2 / *Example 2*

```
If ( variable < Numero )

      do something

End If
-------------------------------------
```

En este ejemplo evaluamos si un valor es menor a otro, para ejecutar la instrucción. != distinto

In this example we check if a variable is lesser than a number to execute the instruction.

Ejemplo 3 / *Example 3*

```
If ( variable1 != variable2 )

      Ejecutar instrucciones

End If
```

En este ejemplo evaluamos si dos variables son distintas, en cuyo caso se ejecuta la instrucción.

In this example we check if a variable is different than a number to execute the instruction.

Tipos de Condicionales / *Types of Conditionals*

=	igual a / *equals*
>	mayor que / *greater than*
<	menor que / *less than*
>=	mayor o igual que / *greater than or equal to*
<=	menor igual que / *less than or equal to*
Not	diferente de / *not equal*
Or	o / *or*
And	y / *and*

Ejemplo / *Example*

```
If  n = 20

      Print ("La condicion se cumple");

End If
```

Ejercicio Guiado # 2 *Sample Code*
Ejercicio de seudo-código con condicionales *PseudoCode using Conditionals*
Usar la variable del clima para ejecutar instrucciones sobre la forma de vestirse *Use weather as a variable get dressed*

Ejemplo / *Example*

```
=========================================================
 BREAK

 if (estado = "hambriento/hungry" Or estado = "cansado/tired") then

      almorzar()

 end if

=========================================================
```

Ciclos / *Loops*

Al escribir un programa, a veces es necesario que una determinada instrucción se repita un cierto número de veces (como por ejemplo crear las columnas de un edificio, o sumar un número de variables para encontrar un total). Los ciclos nos permiten formular este tipo de instrucciones repetitivas que se ejecutan hasta que determinada condición se cumple.

PseudoCo When writing a script, we sometimes need a certain instruction to be performed a certain number of times (for example creating a series of columns in a building, or doing some kind o operation to all the members in an array of variables)*de using Conditionals*

```
For ( una determinada condición )

      Ejecutar instrucción

End For
```

Ejemplo / *Sample Code*

```
Dim suma

      For n = 0 to 20

            Suma = Suma + n

      Next
```

Algoritmos / *Algorithms*

Un algoritmo es una serie finita de pasos para la solución de un problema.

An algorithm is a set of well defined rules for the solution of a problem in a finite number of steps

McGraw-Hill Dictionary of Physics and Mathematics. New York: McGraw-Hill, 1978. (Traducción no oficial)

McGraw-Hill Dictionary of Physics and Mathematics. New York: McGraw-Hill, 1978.

Una receta matemática para la solución de un problema.

A recipe for solving a mathematical problem.

Peat, D. From Certainty to Uncertainty: The story of Science and the Ideas in the Twentieth Century, Joseph Henry Press, 2002 (Traducción no oficial).

Peat, D. From Certainty to Uncertainty: The story of Science and the Ideas in the Twentieth Century, Joseph Henry Press, 2002

Podemos usar los temas vistos hasta ahora para la creación de un algoritmo. A continuación describiremos una serie de estrategias útiles para mantener la cordura al escribir un programa. Este taller le da una importancia capital a estas estrategias.

We can use all of the topics above to create an algorithm. Here are some useful strategies to keep sanity while coding. This workshop will make strong emphasis in these strategies.

Estrategias / *Strategies*

Estrategia #1
Seudo-código

Antes de comenzar a escribir un programa es muy útil tener una descripción clara y detallada de la solución. Una descripción de un algoritmo en lenguaje natural, parecida a nuestro ejercicio del clima, es lo que llamamos seudo-código, y es una manera eficiente de evaluar la lógica de una solución, antes de escribir el programa en la sintaxis propia del lenguaje.

Strategy #1
Pseudocoding

Before starting to write actual code it very is useful to have a clear and structured description of the solution. A description of an algorithm done using natural language, similar to the weather exercise above, is what we call a pseudocode, and it provides a way of evaluating the logic of the solution before dealing with syntax issues.

Estrategia #2
Comentarios

Escribir comentarios en lenguaje natural es una estrategia muy útil que facilitará editar el código en futuras oportunidades, o permitir que otra persona trabaje en él.

Strategy #2
Commenting

Writing comments in natural language within the code is a good practice that will make it easier to edit the code later, or give it to someone else.

Ejemplo

```
'En este ciclo estoy hallando la suma
de los elementos del arreglo.

Dim total = 0
For  m = 0 to 30
      Total = Total + arrayOfValues(m)
Next
```

Example

```
'In this for loop I'm finding the
total sum of the contents of
arrayOfValues

Dim total = 0
For m = 0 to 30
      Total = Total +
arrayOfValues(m)
Next
```

Estrategia #3
Debugging

Significa encontrar y solucionar los errores que hacen que un programa no funcione como es preciso. Para encontrar estos errores es aconsejable seguir los siguientes pasos.

Imprimir sistemáticamente las variables del programa.

- Para comprobar que las variables se comportan como se espera
- Para verificar donde ocurre el error
- Para verificar que determinada función está siendo ejecutada

Ejemplo

```
Rhino.Print "Aca estoy"
Rhino.Print "Adentro de determinada
función"
```

Strategy #3
Debugging

Print statements and printing variables:

- *for checking the value of variables*
- *for verifying location where ERROR occurs in code*
- *for verifying if function is executing*

example:

Rhino.Print "I am here…"
Rhino.Print "Inside 'some function'…."

Funciones / *Functions*

Una función es una pieza de código que ejecuta una tarea determinada. Una vez creadas estas piezas de código pueden ser llamadas desde otras secciones del código. Es aconsejable encapsular tareas repetitivas dentro de funciones; esto permite tener iun código más breve y eficiente, y por lo tanto, más fácil en términos de 'debugging'.

Ejemplo de funciones anidadas

```
Function fiesta()
      Reunirme con amigos
      Call bailar()
End function

Function bailar()
      Busca una pareja
      Acercarte a la pareja
      Pídele a tu pareja bailar
      Tú y tu pareja se mueven al ritmo
de la música
End function
```

A function is a closed body of code that executes a specific task. Functions are usually instigated from other locations within the code structure. Creating functions helps in the housekeeping of code for the purpose of debugging and delegating little areas of code to do specific tasks.

Example:

```
Function party()
      Mingle will friends
      Call dance()
End function

Function dance()
      Look for a partner
      Approach your partner
      Ask your partner to dance
      You and partner move to the beat of the
music
End function
```

Manipulación Geométrica / *Geometry manipulation*

RhinoScript permite un control avanzado de geometría a continuación una mirada a las herramientas que permiten este control.

Qué son los parámetros U, V?

- Las superficies construidas en Rhino no son referenciadas internamente por medio del sistema cartesiano x,y,z.
- Las superficies NURBS se referencian al espacio U,V, que es local en lugar de global.
- Para encontrar un punto en una superficie, utilizamos las coordenadas locales U y V. (Por ejemplo: Rhino.evaluateSurface(surface,Array(3,5)))
- Una manera sencilla de visualizar los parámetros U y V es mirar las isocurvas.
- Esto nos permite recorrer ordenadamente una superficie. Ver ejemplo.

- *Introduction to U,V local coordinates for objects*
- *Surfaces are not referenced internally by Cartesian pts (x,y,z)*
- *Nurbs surfaces are parameterized to a local U,V space*
- *To find a point on a surface, we pass values to Rhino for U,V parameters (example: Rhino.evaluateSurface(surface, Array(3,5)))*
- *An easy way to visualize U,V is to look at the isocurves*

How to iterate across a surface (see script for complete example):

Ejemplo / *Sample Code*

```
for i = 0 to uMax

for j = 0 to vMax
            'obtener los parámetros u de la superficie
            'get the uparam on the surface
            uParam = i*(uDomain(1)-uDomain(0))/uMax

            'obtener los parámetros v de la superficie
            'get the vaparam on the surface
            vParam = j*(vDomain(1)-vDomain(0))/vMax

            'Encontrar el punto a partir de las coordenadas U, V
            'find the point at the U and V parameters
            pt = Rhino.EvaluateSurface(srf, Array(uParam, vParam))

            'Añadir un punto en ese lugar
            'add the point
            call Rhino.addPoint(pt)

            'Podrías añadir otras geometrias en este punto?
            'could you draw other geometries here?
      next
next
```

Usando curvas isoparamétricas / *Using isoparametric curves*

```
Rhino.ExtractIsoCurve (strObject , arrParameter, intDir)
```

Cuando sería útil una isocurva? / *When would an isocurve be valuable?*

Funciones comunes para crear isocurvas / *Common functions to build surfaces:*

AddLoftSrf	'hace 'loft' en una serie de curvas *'lofts over a set of curves*
AddEdgeSrf	'crea una superficie a partir de 2, 3, 4 curvas *'makes a surface from 2,3,4 edge curves*
AddSrfPtGrid	'Crea una superficie a partir de una serie de puntos *'makes a surface from a grid of points*

Ahora que sabemos como extraer información de una superficie, podemos crear nuestra propia superficie?

Now that you know how to extract information from a surface (points and isocurves) can you build your own surface?