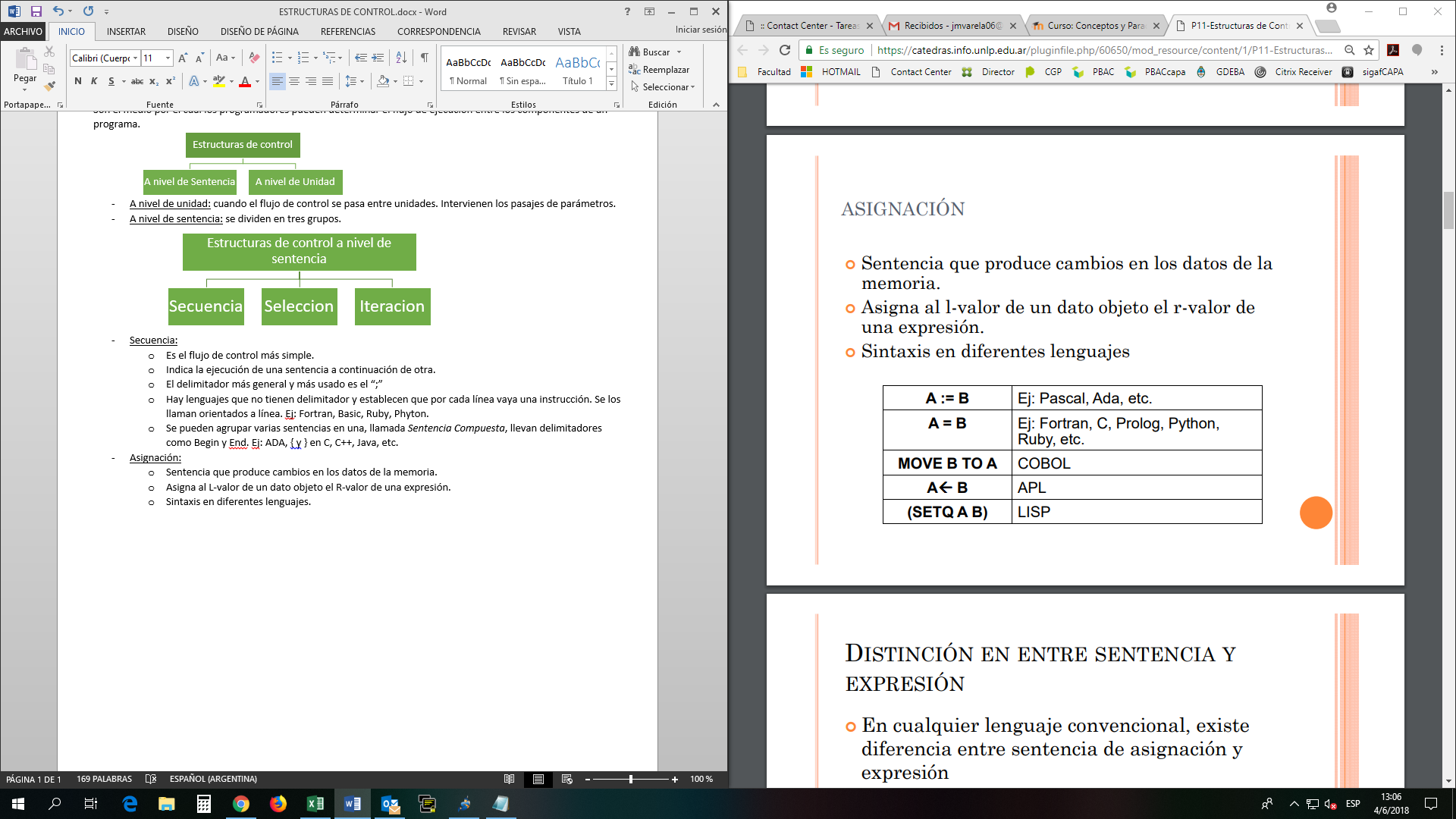
**ESTRUCTURAS DE CONTROL**

Son el medio por el cual los programadores pueden determinar el flujo de ejecución entre los componentes de un programa.

* A nivel de unidad: cuando el flujo de control se pasa entre unidades. Intervienen los pasajes de parámetros.
* A nivel de sentencia: se dividen en tres grupos.
* Secuencia:
  + Es el flujo de control más simple.
  + Indica la ejecución de una sentencia a continuación de otra.
  + El delimitador más general y más usado es el “;”
  + Hay lenguajes que no tienen delimitador y establecen que por cada línea vaya una instrucción. Se los llaman orientados a línea. Ej: Fortran, Basic, Ruby, Phyton.
  + Se pueden agrupar varias sentencias en una, llamada *Sentencia Compuesta*, llevan delimitadores como Begin y End. Ej: ADA, { y } en C, C++, Java, etc.
* Asignación:
  + Sentencia que produce cambios en los datos de la memoria.
  + Asigna al L-valor de un dato objeto el R-valor de una expresión.
  + Sintaxis en diferentes lenguajes.



* Distinción entre sentencia y expresión:
  + En cualquier lenguaje convencional, existe diferencia entre sentencia de asignación y expresión.
  + En otros lenguajes tales como C definen la sentencia de asignación, como una expresión con efectos laterales.
  + Las sentencias de asignación devuelven valores.
  + Evalúa de derecha a izquierda en C
  + Ejemplo a=b=c=0;
  + If (i=30) printf(“es verdadero”)
  + La mayoría de los lenguajes de programación requieren que sobre el lado izquierdo de la asignación aparezca un l-valor. C permite cualquier expresión que denote un l-valor.
* Selección:
  + Esta estructura de control permite que el programador pueda expresar una elección entre un cierto número posible de sentencias alternativas.
  + Evolución:

🡪 If lógico de Fortran

* + - * If (condición lógica) sentencia

Si la condición es verdadera ejecuta la sentencia.

🡪 If then else de Algol

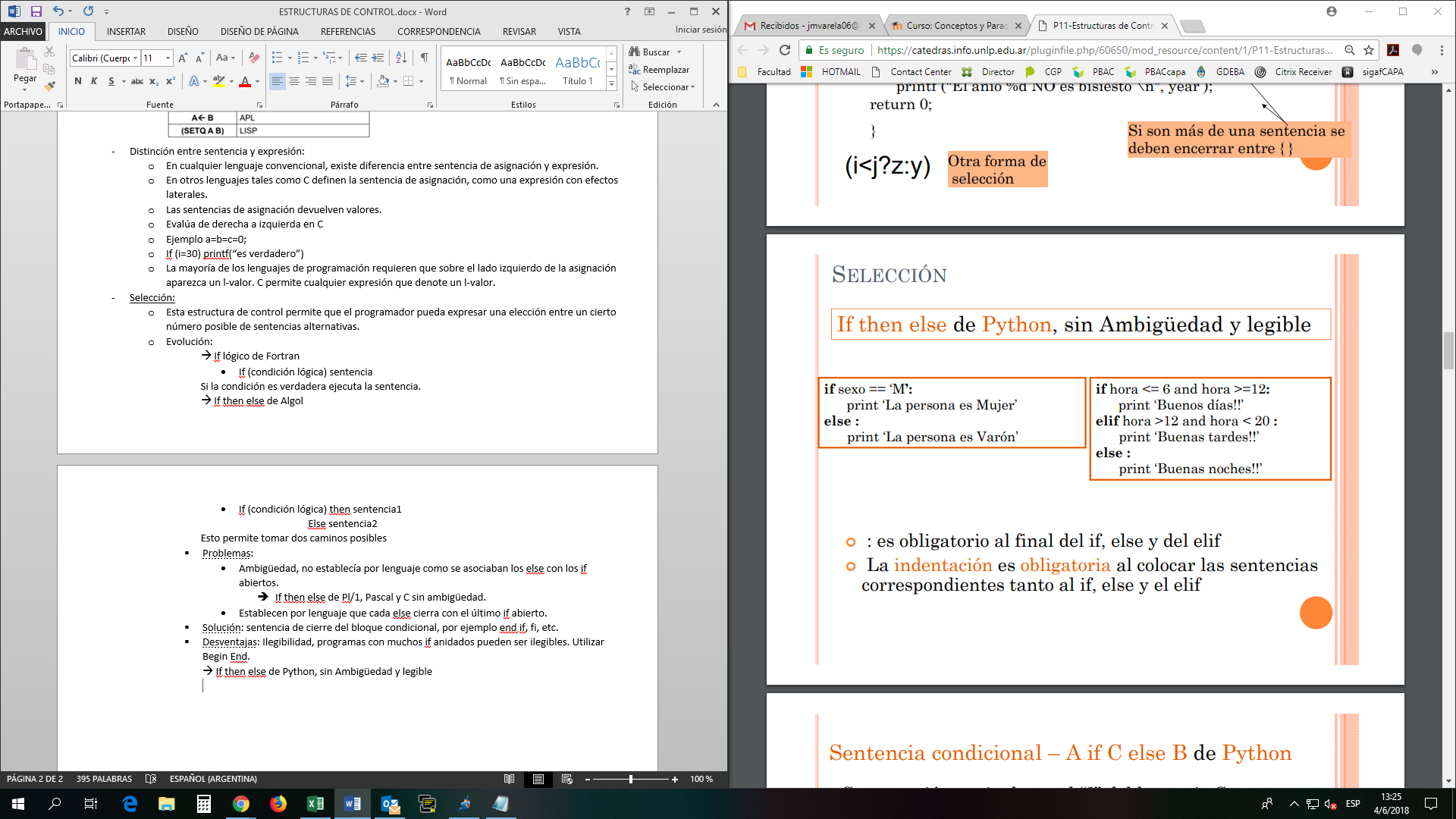
* + - * If (condición lógica) then sentencia1

Else sentencia2

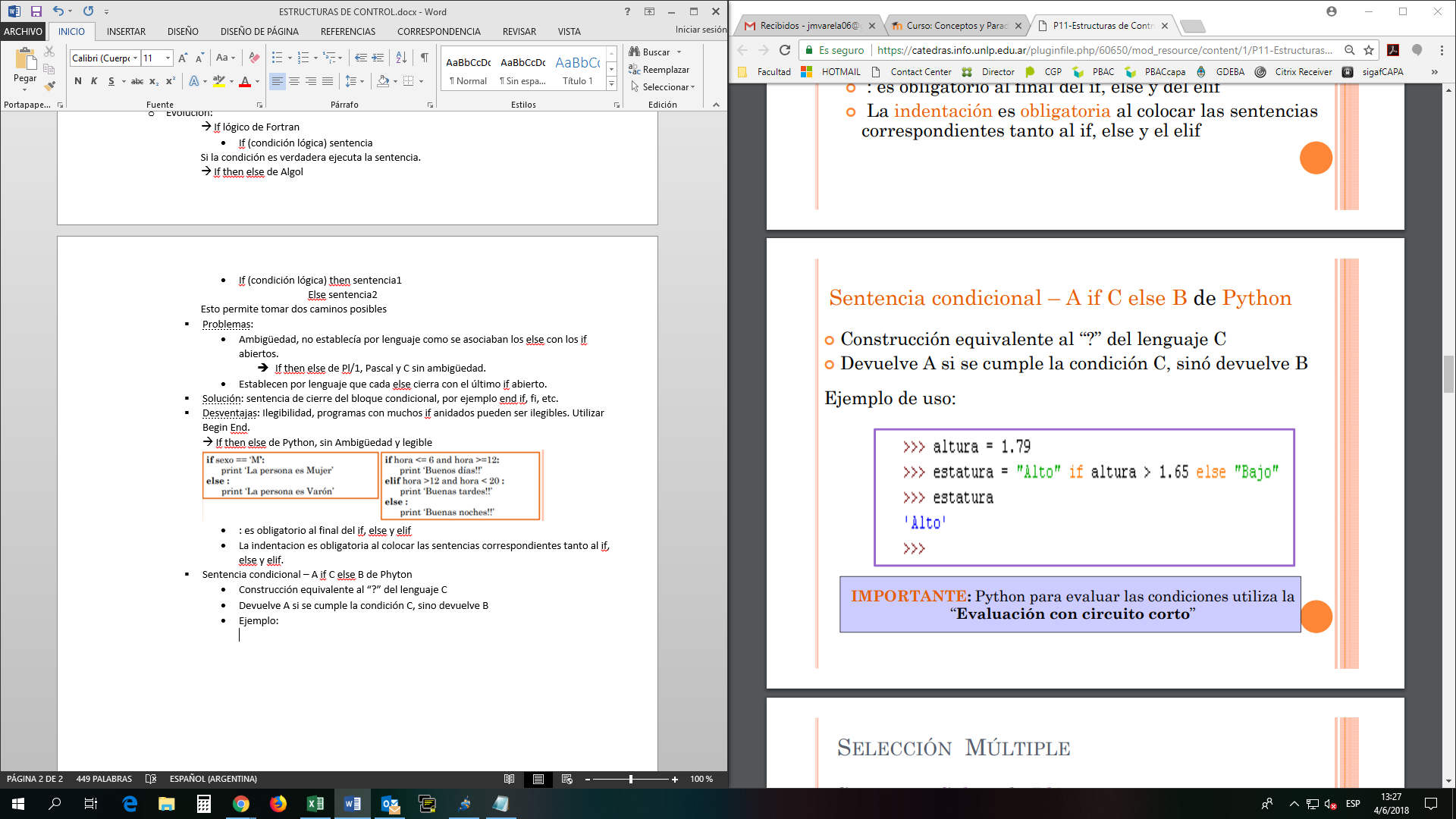
Esto permite tomar dos caminos posibles

* + - Problemas:
      * Ambigüedad, no establecía por lenguaje como se asociaban los else con los if abiertos.
        + If then else de Pl/1, Pascal y C sin ambigüedad.
      * Establecen por lenguaje que cada else cierra con el último if abierto.
    - Solución: sentencia de cierre del bloque condicional, por ejemplo end if, fi, etc.
    - Desventajas: Ilegibilidad, programas con muchos if anidados pueden ser ilegibles. Utilizar Begin End.

🡪 If then else de Python, sin Ambigüedad y legible



* + - * : es obligatorio al final del if, else y elif
      * La indentacion es obligatoria al colocar las sentencias correspondientes tanto al if, else y elif.
    - Sentencia condicional – A if C else B de Phyton
      * Construcción equivalente al “?” del lenguaje C
      * Devuelve A si se cumple la condición C, sino devuelve B
      * Ejemplo:



* Selección Multiple:
  + Sentencia Select de Pl/1: Pl/1 incorpora la sentencia de selección entre dos o más opciones.

Select

When (a) sentencia1;

When (b) sentencia2;

……

Otherwise sentencia n;

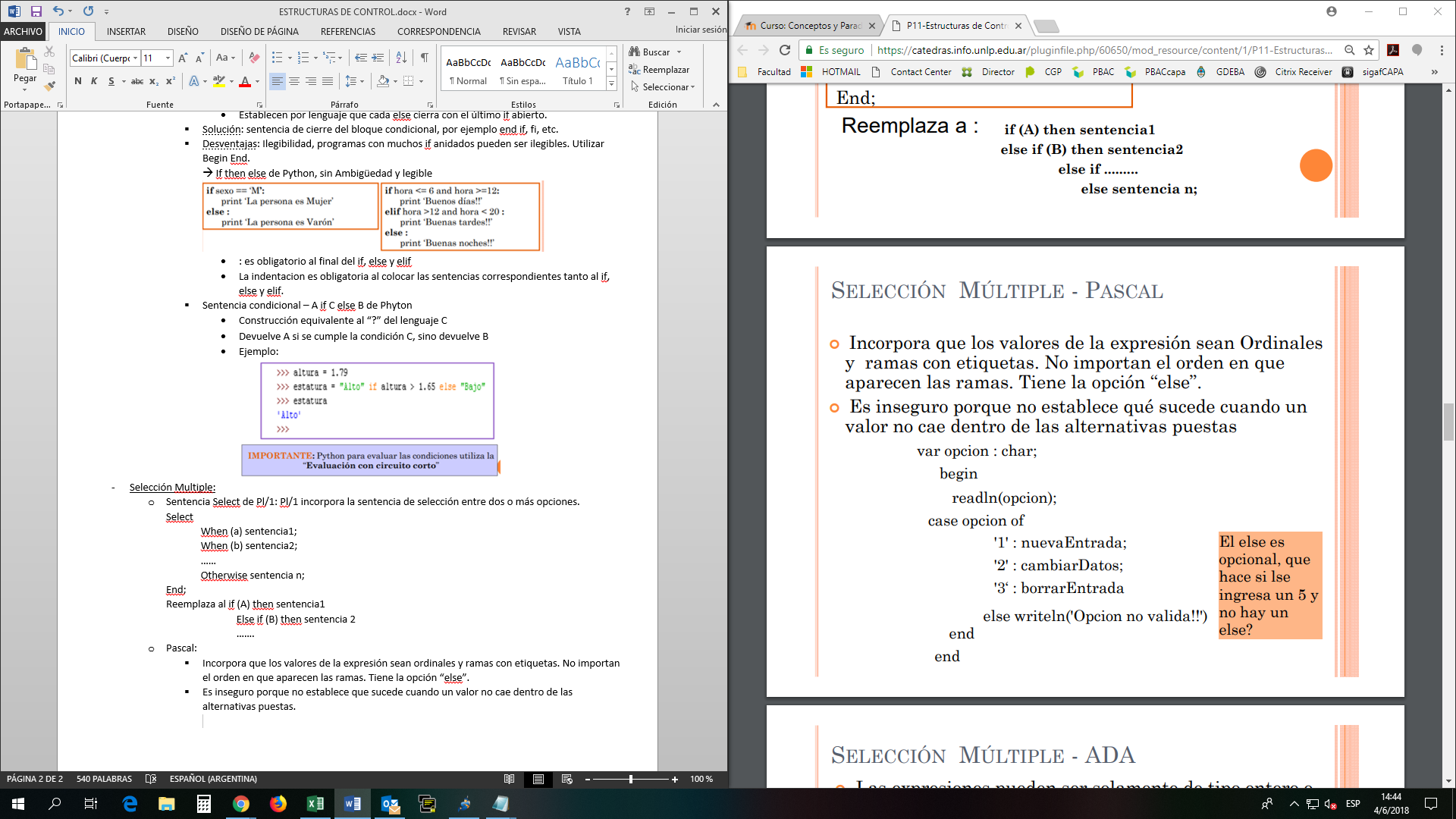
End;

Reemplaza al if (A) then sentencia1

Else if (B) then sentencia 2

…….

* + Pascal:
    - Incorpora que los valores de la expresión sean ordinales y ramas con etiquetas. No importan el orden en que aparecen las ramas. Tiene la opción “else”.
    - Es inseguro porque no establece que sucede cuando un valor no cae dentro de las alternativas puestas.



* + ADA:
    - Las expresiones pueden ser solamente de tipo entero o enumerativo.
    - En las selecciones se debe estipular todos los valores posibles que puede tomar la expresión.
    - Tiene la cláusula Others que se puede utilizar para representar a aquellos valores que no se especificación explícitamente.

Si NO se coloca la rama para un posible valor o si NO aparece la opción Others en esos casos, no pasara la compilación.

* + - Ejemplo 1:

Case operador is

When ‘+’ => result:=a + b;

When ‘-‘ => result:=a – b;

When Others => result:=a \* b;

End case

La cláusula others se debe colocar porque las etiquetas de las ramas NO abarcan todos los posibles valores de Operador.

* + - Ejemplo 2:

Case hoy is

When MIE..VIE => Entrenar\_duro; -- rango

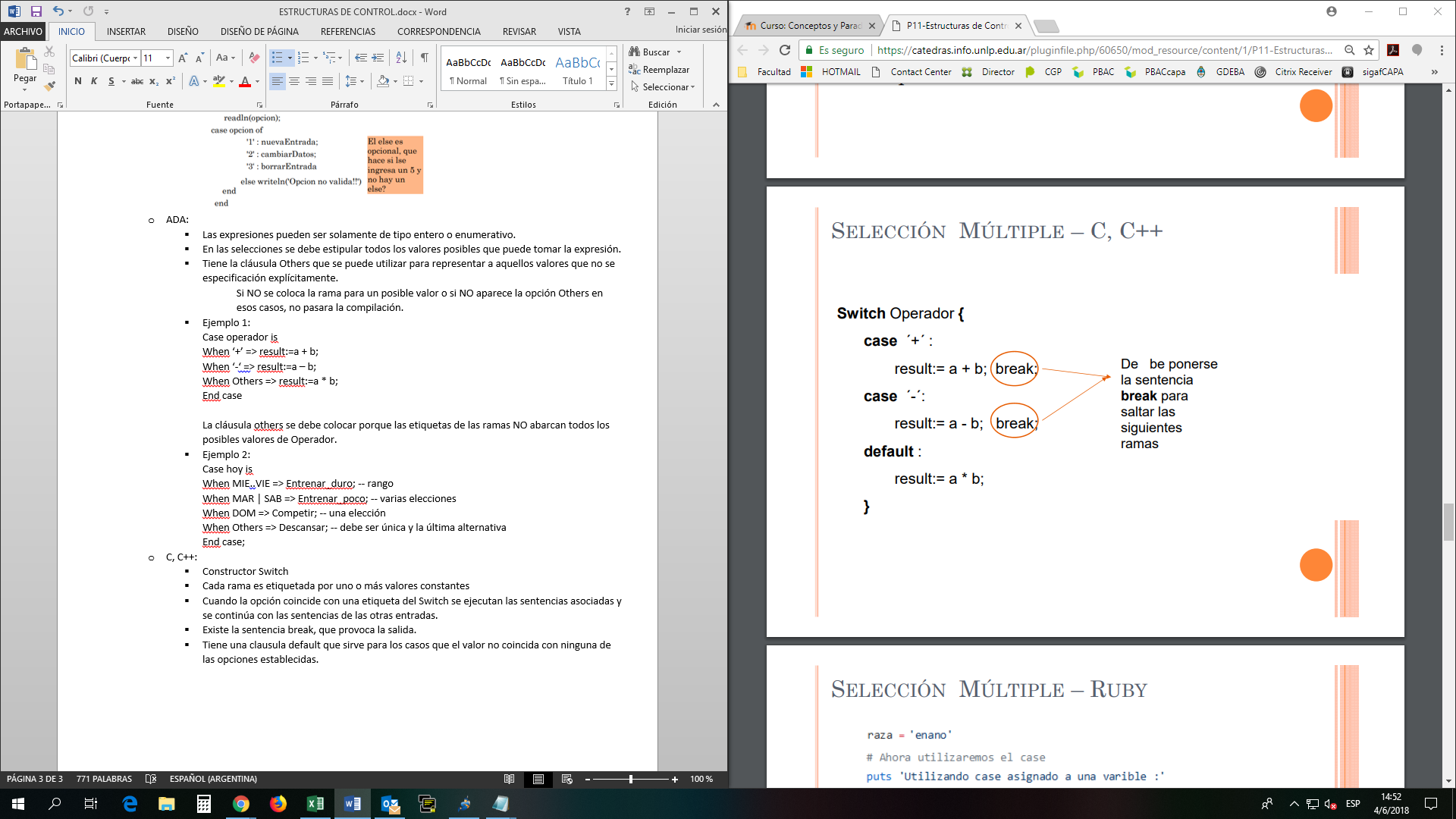
When MAR | SAB => Entrenar\_poco; -- varias elecciones

When DOM => Competir; -- una elección

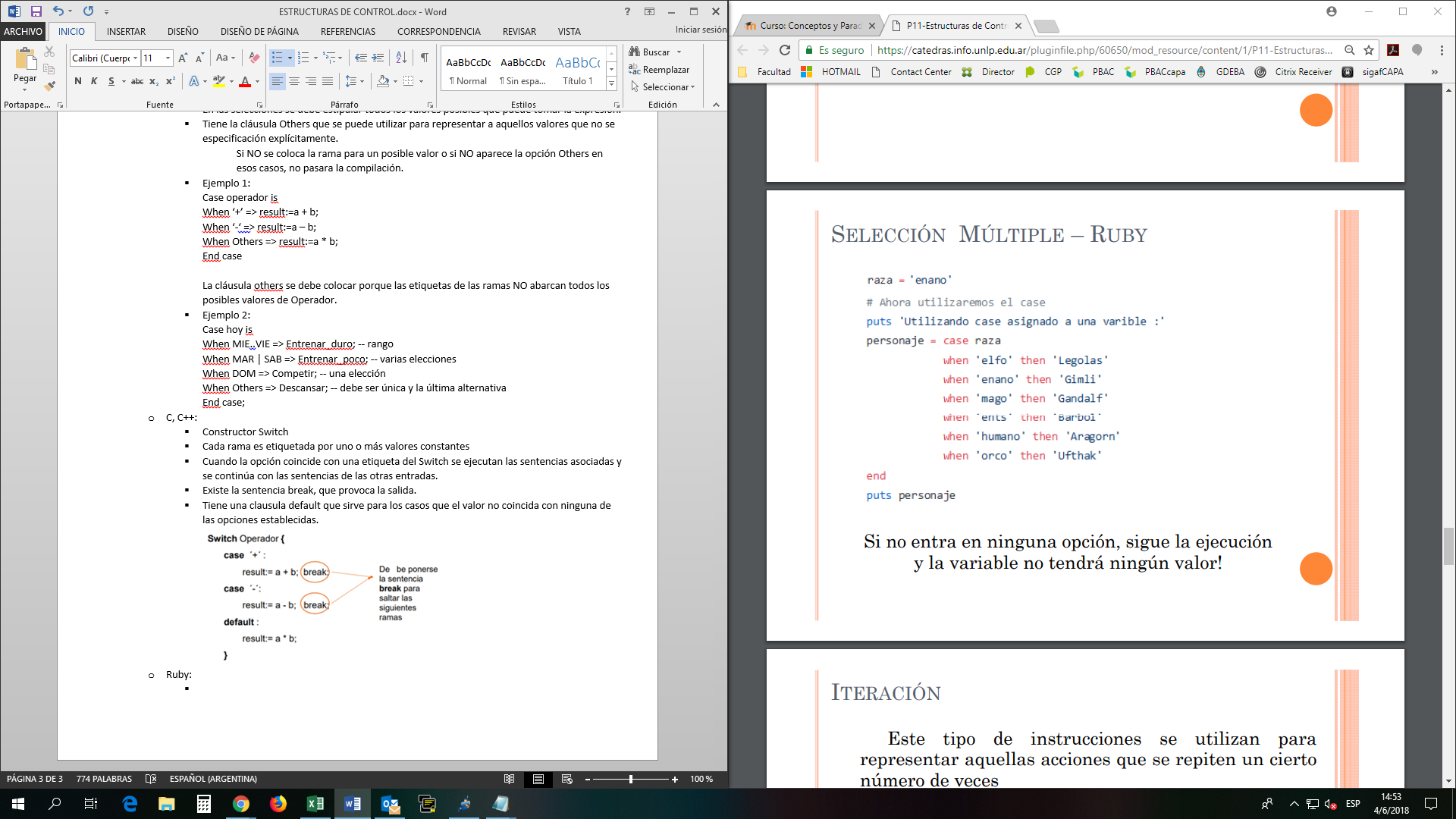
When Others => Descansar; -- debe ser única y la última alternativa

End case;

* + C, C++:
    - Constructor Switch
    - Cada rama es etiquetada por uno o más valores constantes
    - Cuando la opción coincide con una etiqueta del Switch se ejecutan las sentencias asociadas y se continúa con las sentencias de las otras entradas.
    - Existe la sentencia break, que provoca la salida.
    - Tiene una clausula default que sirve para los casos que el valor no coincida con ninguna de las opciones establecidas.



* + Ruby:



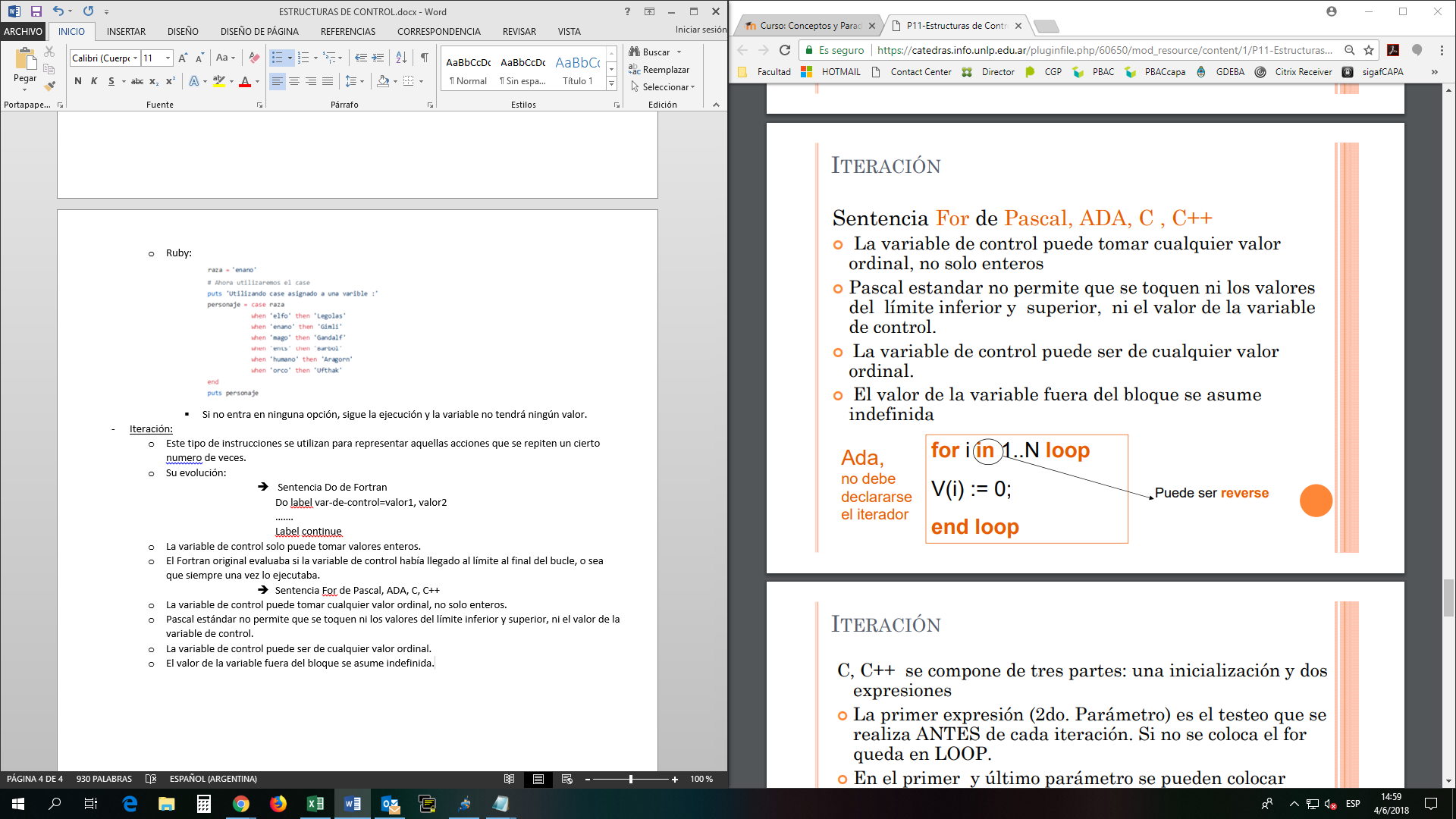
* + - Si no entra en ninguna opción, sigue la ejecución y la variable no tendrá ningún valor.
* Iteración:
  + Este tipo de instrucciones se utilizan para representar aquellas acciones que se repiten un cierto número de veces.
  + Su evolución:
    - * + Sentencia Do de **Fortran**

Do label var-de-control=valor1, valor2

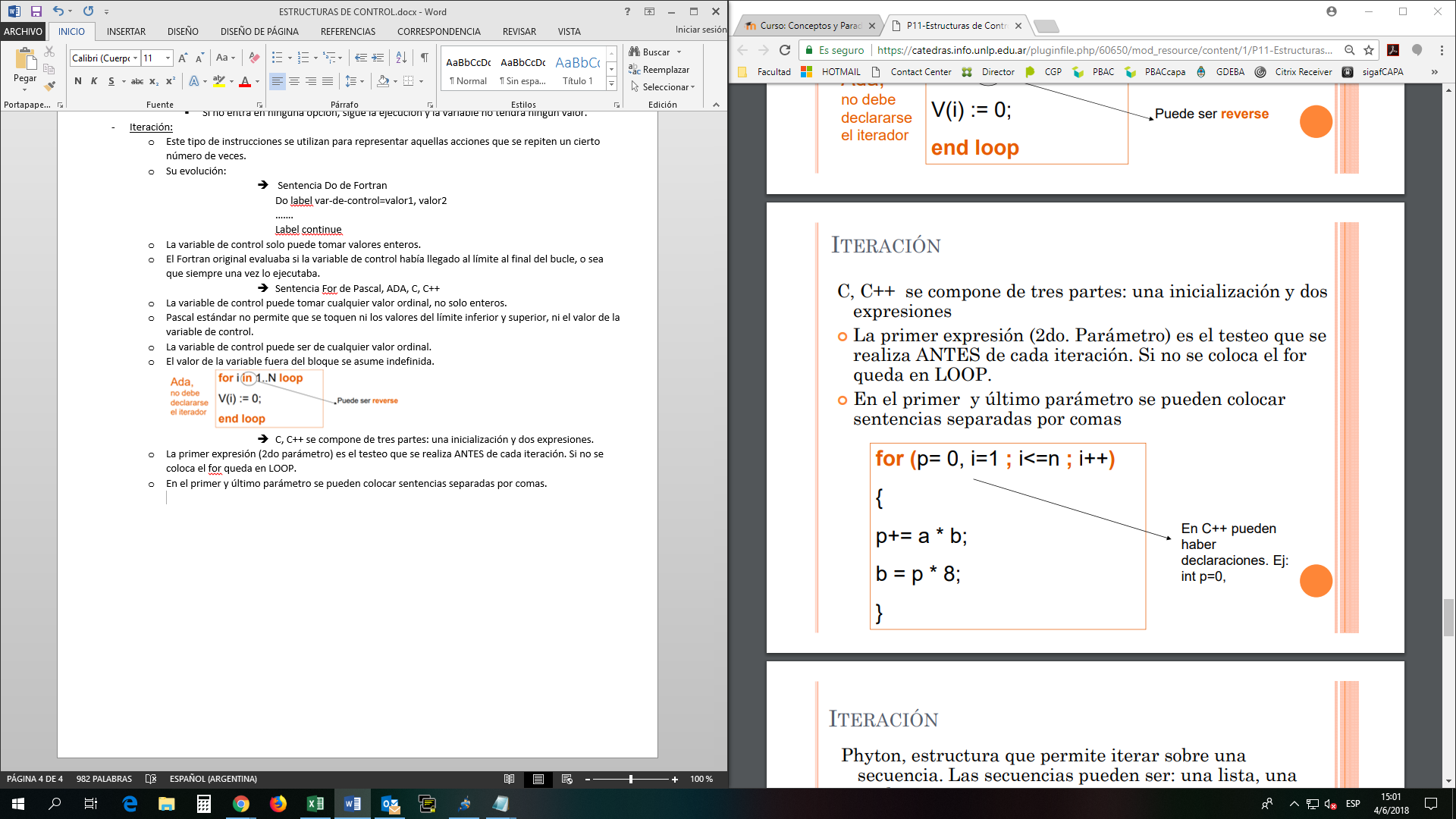
…….

Label continue

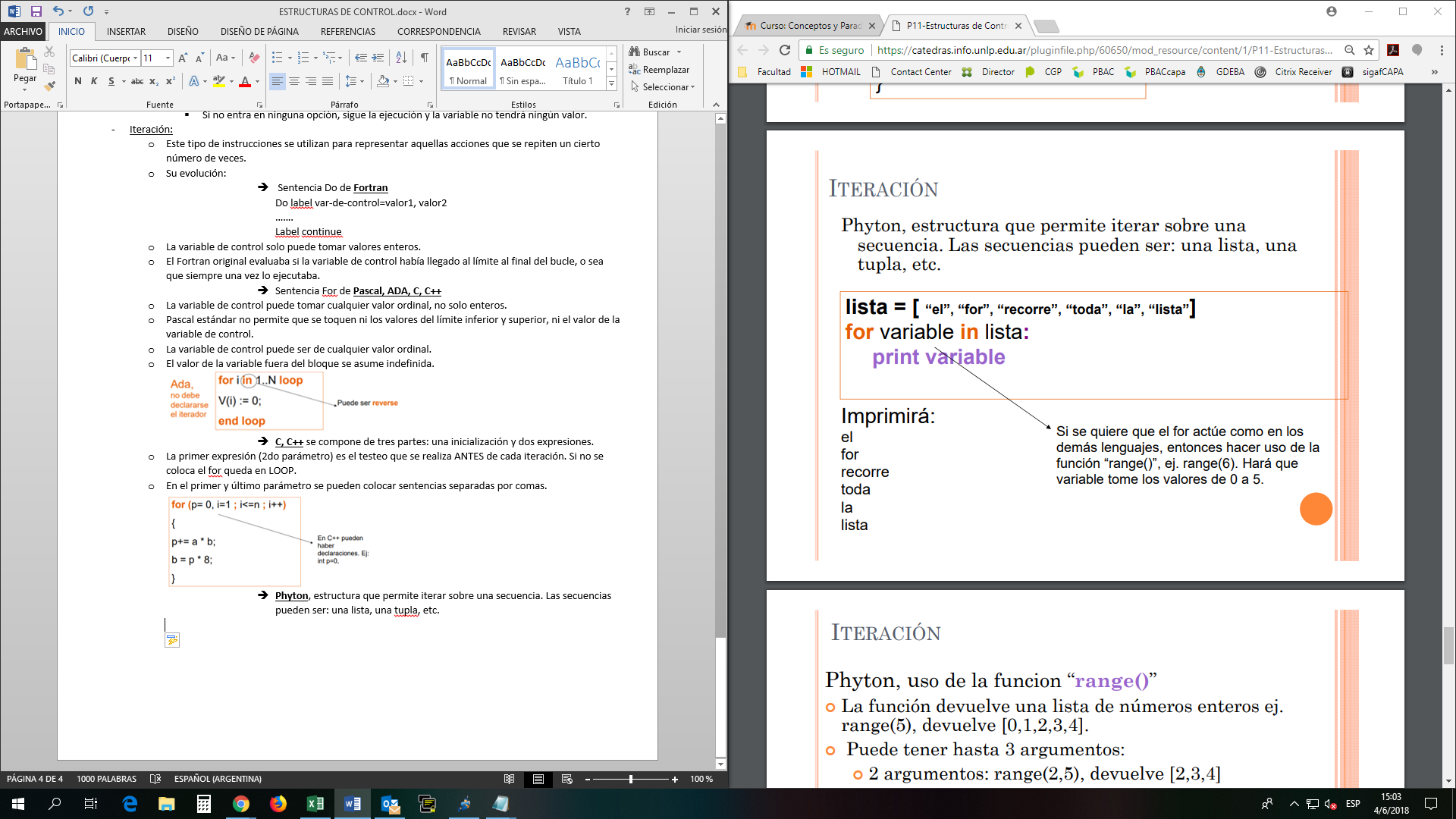
* + La variable de control solo puede tomar valores enteros.
  + El Fortran original evaluaba si la variable de control había llegado al límite al final del bucle, o sea que siempre una vez lo ejecutaba.
    - * + Sentencia For de **Pascal, ADA, C, C++**
  + La variable de control puede tomar cualquier valor ordinal, no solo enteros.
  + Pascal estándar no permite que se toquen ni los valores del límite inferior y superior, ni el valor de la variable de control.
  + La variable de control puede ser de cualquier valor ordinal.
  + El valor de la variable fuera del bloque se asume indefinida.



* + - * + **C, C++** se compone de tres partes: una inicialización y dos expresiones.
  + La primer expresión (2do parámetro) es el testeo que se realiza ANTES de cada iteración. Si no se coloca el for queda en LOOP.
  + En el primer y último parámetro se pueden colocar sentencias separadas por comas.



* + - * + **Phyton**, estructura que permite iterar sobre una secuencia. Las secuencias pueden ser: una lista, una tupla, etc.



* + - * + **Phyton**, uso de la función “range()”
  + La función devuelve una lista de números enteros ej. range(5), devuelve [0,1,2,3,4].
  + Puede tener hasta 3 argumentos:
    - 2 argumentos: range(2,5), devuelve [2,3,4]
    - 3 argumentos: range(2,5,2), devuelve [2,4]

Esa función range() da la posibilidad de simular la sentencia FOR de otros lenguajes

For i in range (valor-inicial, valor-final + 1): acciones

Ejemplo: for vuelta in range(1,10):

Print(“vuelta “+str(vuelta))

* Iteración: While
  + Estructura que permite repetir un proceso mientras se cumpla una condición.
  + La condición se evalua antes de que se entre al proceso.
    - En pascal:
      * While condición do sentencia;
    - En C, C++:
      * While (condición) sentencia;
    - En ADA:
      * While condición sentencia

End loop;

* + - En Phyton:
      * While condición:

Sentencia1

Sentencia2

…

Sentencia n

* Iteración: Until
  + Estructura que permite repetir un proceso mientras se cumpla una condición.
  + La condición se evalúa al final del proceso, por lo que por lo menos una vez el proceso se realiza.
    - En Pascal:
      * Repeat

Sentencia

Until condición;

* + - En C, C++:
      * Do sentencia;

While (condición);

* Iteración: Loop
  + ADA tiene una estructura iterativa

Loop

…

End loop;

* + De este bucle se sale normlamente, mediante una sentencia “exit when” o con una alternativa que contenga una clausula “exit”.

Loop

…

Exit when condición;

…

End loop;