Programación con Python nivel avanzado

Estructuras de datos y funciones

Introducción

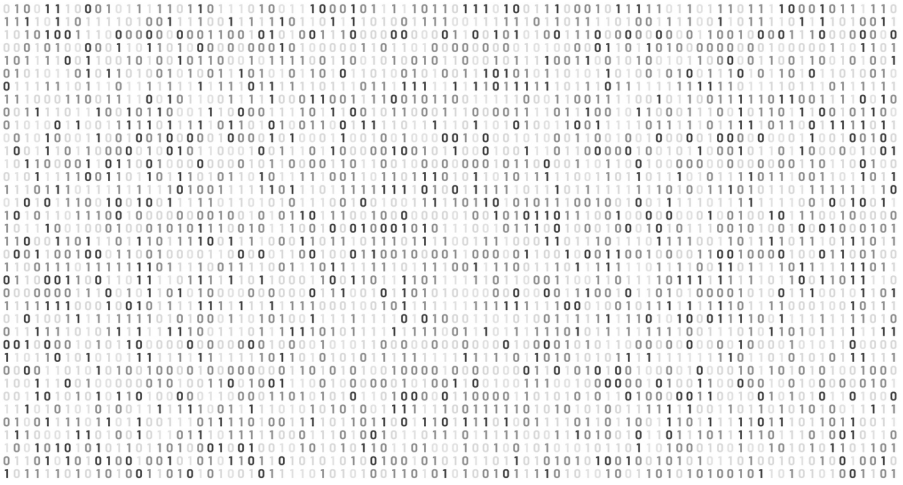
En la década de los 80's con la adaptación del uso de cómputo a nivel de computadora personal comenzó la evolución del trabajo y los elementos que se consideraban fundamentales en la programación, puesto que los desarrolladores comenzaron a crear métodos que ayudarían a los lenguajes a ser más versátiles en la toma de decisiones y generar cambios posteriores en las líneas de código. Python es uno de esos lenguajes creados para este tipo de tareas. En esta experiencia educativa se abordarán las estructuras básicas de Python para almacenar y dar tratamientos a los datos de manera temporal mediante las listas y diccionarios, también se trabajarán con los conceptos de segmentación y modularidad lo que permite fraccionar la funcionalidad del código en partes más pequeñas, la unidad fundamental de esta estrategia radica en las funciones.

Los lenguajes de programación permiten crear iteraciones de operaciones las cuales hacen uso del poder de la repetición y del cálculo computacional, son usados comúnmente por los programadores para hacer que las tareas o cálculos repetitivos se realicen a través de una estructura automatizada, ya sea basada en una secuencia que se sabe de antemano cuantas veces se repetirá, o bien, cuando se existe una condición que establezca el cese o la repetición misma.

Explicación

Conocer y utilizar la sintaxis de las estructuras para almacenar y acceder a elementos

Las listas en Python tienen varias de estructuras para almacenar datos, una de ellas son las llamadas listas, estas son una estructura muy utilizada y es muy versátil, almacenan información que puede llamar elementos, estos, posteriormente, se pueden extraer o incluso agregar a las listas antes ya creadas. Su sintaxis de construcción se caracteriza porque está definida por corchetes y los elementos que las conforman están separados por comas.



En las listas se pueden crear elementos del mismo tipo, o bien, pueden variar, de esta manera también se pueden incluir otras listas dentro de la principal, a esto se le llama **lista heterogénea**, y estas pueden modificar sus elementos haciéndolas mutables.

Para ejemplificar el uso de las listas se tiene el siguiente código:

```
lista = ["Lunes", "Martes", "Miercoles","Jueves", "Viernes"]
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Se pueden acceder a los elementos de las listas mediante su **índice**, es decir que cada elemento tiene, automáticamente, un orden según esté colocado en la lista y Python siempre toma el **0** como primer elemento, es decir, que para acceder a este elemento se debe tomar la lista y agregar su índice.

```
lista = ["Lunes", "Martes", "Miercoles","Jueves", "Viernes"]

print (lista[0])
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Existe una función que devuelve la longitud de la lista, de esta manera se sabrá de cuantos elementos está conformada la lista, esta función es **len()**.

```
lista = ["Lunes", "Martes", "Miercoles","Jueves", "Viernes"]

len(lista)
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

A través de los índices es posible cambiar los elementos de una lista de forma rápida y dinámica, es decir, que puedes seleccionar el elemento por el cual se va a remplazar.

```
lista = ["Lunes", "Martes", "Miercoles","Jueves", "Viernes"]

lista[2] = "Domingo"
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Las listas tienen varios **métodos** para la integración de nueva información o incluso para eliminar elementos o de ellas, o bien, toda la lista de acuerdo con la documentación oficial de Python Software Foundation (s.f.):

- **append()**: se anexa un valor al final de la lista.
- **count()**: método que devuelve la cantidad de elementos que posee una lista.
- **extend(iterable)**: extiende una lista agregando la totalidad de datos de un iterable.
- **insert(i,x)**: inserta un dato x en la lista, a partir del índice i.
- **pop(i)**: quita el ítem dato de la lista i se refiere al índice o posición en caso de dejar vacío el método toma el último elemento por defecto.
- **remove(x)**: este método recibe como argumento (x) un elemento, y borra su primera aparición en la lista.
- **reverse()**: Invierte el orden de los elementos de una lista.
- **sort()**: se ordenan los elementos de una lista.

Tuplas. Otra parte fundamental en la estructura de datos son las tuplas, estas también son de tipo secuencia con la gran diferencia de que estas no se pueden modificar, en otras palabras, son estructuras de solo lectura y su estructura se define con paréntesis.

Explicación

```
deportes = ('Futbol', 'Basquetbol','Tenis', 5)
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Diccionarios. Este concepto en la estructura de datos es uno de los elementos más importantes. Estos los diccionarios se definen como una relación uno a uno entre claves y valores, es decir, es una estructura donde los elementos que la componen estén formados por una dupla de valores, una llave **(key)** y el valor **(value)**. Los objetos creados en los diccionarios son mutables, es decir, que pueden cambiarse, su estructura se define entre corchetes y cada elemento se integra por **{key: value}** y se pueden agregar tantos elementos como sean necesarios en el diccionario. Otra forma de decirle a estos diccionarios es ***mapping*** o mapeo de elementos.

```
midiccionario = {
    "KEY001" : "Iron Man",
    "KEY002" : Tema2_Python avanz_22 sep_ET.docx
    "KEY004" : False,
    "KEY005" : [1,2,3,4]
}
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Los objetos de tipo diccionario permiten realizar **operaciones** usando operadores integrados en el intérprete de Python para su tratamiento:

- **Acceder al valor clave**: esta operación permite acceder a un valor específico del diccionario mediante su clave.

- **Asignar valor a clave**: esta operación le permite asignar el valor específico del diccionario mediante su clave.

```
midiccionario = {
    "KEY001" : "Iron Man",
    "KEY002" : 3.14169,
    "KEY003" : False,
    "KEY004" : [1,2,3,4]
}
#Asignación de un valor a una clave
midiccionario["KEY002"] = 2.0910
#Acceder al valor contenido en un diccionario por clave
print(midiccionario["KEY002"])
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

También los **diccionarios tienen métodos** con los que se pueden agregar y borrar elementos dentro de estos, estos son algunos de ellos:

Conjuntos. Otro tipo de estructura de datos en Python. Un conjunto es una colección no ordenada y sin elementos repetidos, ya que uno de sus usos básicos es la verificación de pertenencia y la eliminación de entradas duplicadas. Existen dos tipos de conjuntos que se establecen desde su codificación:

1. **Set:** son conjuntos mutables, sin orden y no contiene duplicados.

2. **Frozenset:** son conjuntos inmutables, sin orden, no contiene duplicados.

```
set ([4.0, 'Carro', True])

frozenset ([4.0, 'Carro', True])
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Aprende a utilizar funciones en Python

Una **función** es un bloque de código el cual contiene diferentes elementos que crean un conjunto de acciones para determinar un trabajo en específico, llevan un nombre y esto permite que sean reutilizables, posteriormente, haciendo que el código sea más legible.

Entre sus ventajas está la **modularización**, palabra del anglicismo informático que hace referencia a la división de un código en partes más pequeñas y reutilizables que permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado. Dentro de una función se puede escribir todo tipo de código, el cual, en algunas ocasiones puede ser utilizado para devolver algún resultado o simplemente para ejecutar una acción en específico. Las funciones te permiten desarrollar programas más simples, fáciles de entender, de probar y de corregir, en caso de ser necesario.

Esta es una forma fácil de escribir una función:

```
def nombre_de_funcion():
    pass
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

En el ejemplo anterior, la primera línea muestra la utilización de la palabra reservada **def**, la cual le indica a Python que se comenzará a describir una función. A continuación, se coloca el nombre de dicha función seguida de paréntesis cerrados y el símbolo de dos puntos. La segunda línea esta la instrucción **pass** que significa saltar.

```
def nombre_de_funcion():
    print("Hola Tecmilenials")
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

En este caso, el nombre de la función es saludar y el cuerpo está compuesto por la instrucción de imprimir el texto indicado. Una función que solamente se ha declarado no tiene ninguna utilidad hasta que es llamada o invocada, para este caso la sintaxis correcta sería la siguiente:

```
def saludar():
    print("Hola Tecmilenials")

saludar()
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Una función puede recibir diversos **valores de entrada**. Cuando las variables que representan esos valores se utilizan en la declaración de la función se les denominan **parámetros**; entonces, es correcto expresar que, en el siguiente ejemplo, la función **saludar()** recibe el parámetro de entrada “nombre”. El valor que se le pasa a la función durante la llamada a la misma se le denomina **argumento**. De esa forma se puede expresar que, en el siguiente ejemplo, la función **saludar()** es invocada con el argumento “Luciano”.

```
def saludar(nombre):
    print(f"Hola {nombre}, ¿Cómo te ha ido?")

saludar('Luciano')
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Al definir una función, los valores que se reciben se denominan **parámetros**. A continuación, las funciones pueden ser declaradas con múltiples parámetros de entrada, por lo cual, al momento de llamarlas, es necesario utilizar, de igual forma, múltiples argumentos. Esta operación se puede realizar de distintas formas. A continuación, observa algunas de ellas (Matthes, 2019):

Explicación

- **Posicionales**: cuando se invoca a una función, Python hace coincidir los argumentos recibidos con los parámetros declarados. La forma más sencilla de lograr esto es manteniendo el orden en que aparecen los elementos.

- **Palabras claves**: son los argumentos de palabras claves, los cuales consisten en un par nombre-valor que se le pasa a la función a la hora de invocarla. De esta manera se realiza la asociación directa, sin dar pie a confusiones.

- **Llamada sin argumentos**: es posible agregar funciones que no se requieran argumentos para su llamada, dejando, por ende, vació este punto en el código.

- **Argumentos por defecto**: pudiera darse el caso de que al momento de invocar la función omitas alguno de sus argumentos, en esa situación el programa va a generar un error. En situaciones como estas o donde también se puede considerar que el argumento no es obligatorio, una buena práctica es declarar valores por defecto en los parámetros de la función.

Las **funciones tienen varios usos**. Anteriormente, viste la aplicación para mostrar elementos de forma directa, pero también es muy común su utilización en la obtención de resultados que se manejarán, posteriormente, en otras partes del programa. A estos elementos se les conoce como **valores de retorno**.

Una función con **valores de retorno** se puede escribir de la siguiente forma:

```
def restar(x,y):
    return x-y
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

El valor que se desea retornar va precedido por la palabra reservada return.

```
def restar(x,y):
    return x-y

resultado = restar(18,10)
print (resultado)
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Python te permite generar pequeñas funciones anónimas que pueden solucionar un problema particular sin tener que escribir toda una declaración formal. Este tipo de funciones se conocen como **funciones lambda** y son una de las características más poderosas de este lenguaje.

En este caso, la sentencia comienza con la palabra reservada “lambda”, seguida por los argumentos y la expresión que definirá el comportamiento de la función. A continuación, se muestra un ejemplo donde se utiliza una de estas funciones para realizar un sencillo cálculo.

```
disminuir_5_unidades = lambda x: x-6
disminuir_5_unidades(12)

6
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Conocer las bases de la programación estructurada y el concepto DRY *(Don't repeat yourself)*

Silva (2019) menciona que “**Don’t repeat yourself**” (en español, No te repitas) es una especie de mantra, una filosofía que busca eficientizar la lógica en la programación. Su principal objetivo es evitar repeticiones de código innecesarias, una manera de eficientizar la repetición de código innecesaria es auxiliarte por medio del uso de las funciones, así pues, algunas ideas que te pueden ayudar al generar tus funciones buscando maximizar su uso son las siguientes:

- Crea un parámetro en una función que sirva para casi todo lo que se necesita y condiciona la función al nuevo uso.

- Haz uso de la sobrecarga de funciones, es decir, que se pueden crear varias funciones determinando ciertos parámetros y así, el código será menor y el uso será mejor.

Conocer y utilizar los ciclos *while* y *for*

Tal como en otros lenguajes de programación, Python hace uso de operaciones iterativas (repeticiones), los ya tan famosos **bucles de código** que permiten crear ciclos en un periodo de tiempo dentro del programa para poder optimizar el código y conseguir funcionalidades repetitivas y en Python no es la excepción. En Python existen dos tipos de bucles, los que tienen un número de iteraciones no definidas, y los que tienen un número de iteraciones definidas. En otras palabras, las **estructuras de iteración** (ciclos o bucles), te permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición, ya sea verdadera o falsa, o bien, realiza una repetición con base en un número conocido de repeticiones o elementos en una estructura de datos.

Explicación

En **Python los dos tipos de estructuras de iteración** son:

1. **El bucle *while***.
2. **El bucle *for***.

La **estructura while** es una estructura que repetirá un bloque de instrucciones mientras la condición se cumpla.

Ejemplo:

```
suma = 0
#Condicion de entrada al bucle
while suma <=50:
    print(f"La suma total es:{suma} ")
    suma =suma + 10

print("El bucle ha terminado")
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

```
La suma total es:0
La suma total es:10
La suma total es:20
La suma total es:30
La suma total es:40
La suma total es:50
El bucle ha terminado
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Existen bucles infinitos y es que el intérprete de código determina si la sintaxis es correcta o no, pero no determina si un bucle está siendo infinito según corresponda y puede generar errores comunes en el código. Es importante revisar que no se está creando un bucle infinito, el siguiente es un **ejemplo de un bucle infinito**.

```
while True:
    print ("Bucle Infinito")
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Algo no muy corriente en otros lenguajes de programación, pero sí en Python, es el uso de la cláusula ***else*** al final del **while**. En el ejemplo, la sección de código que se encuentra dentro del **else**, se ejecutará cuando el bucle termine, pero solo si lo hace "por razones naturales". Es decir, si el bucle termina porque la condición se deja de cumplir, y no porque se ha hecho uso del ***break***.

```
x = 5
while x > 0:
    x -=1
    print(x) #4,3,2,1,0
else:
    print("El bucle ha finalizado")
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Se observa como si el bucle termina por el **break**, el **print()** no se ejecutará. Por lo tanto, se podría decir que, si no hay realmente ninguna sentencia **break** dentro del bucle, tal vez no tenga mucho sentido el uso del **else**, ya que un bloque de código fuera del bucle cumplirá con la misma funcionalidad.

```
x = 5
while True:
    x -= 1
    print(x) #4, 3, 2, 1, 0
    if x == 0:
        break
else:
    # El print no se ejecuta
    print("Fin del bucle")
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Continuando con el bucle **for**, para Python se puede mencionar que tiene ciertas diferencias con otros lenguajes de programación, es parecido al bucle **while** pero aquí el número de iteraciones está definido de antemano, aquí no se evalúa la condición en cada iteración para decidir si se vuelve a ejecutar o no, sino que en el ciclo **for** solo hay un iterable que decida las veces que se ejecuta el código. En el siguiente ejemplo se observa un bucle for que se ejecuta cinco veces, y donde la **i** incrementa su valor "automáticamente" en uno en cada iteración.

Explicación

```python
for i in range(0, 5):
    print(i)

# Salida:
# 0
# 1
# 2
# 3
# 4
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

En Python se puede iterar, prácticamente todo, como por ejemplo una **cadena**. En el siguiente ejemplo se observa como la **i** va tomando los valores de cada letra.

```python
for i in "Python":
    print(i)

# Salida:
# P
# y
# t
# h
# o
# n
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

También, la **estructura for**, en Python, es aquella que te permite **iterar estructuras de datos** como en el siguiente ejemplo:

Explicación

```python
superheroes = ["Ironman","Batman","Spiderman"]
#Estructrua for que recorrera cada uno de los elementos de la lista superheroes
for encapuchado in superheroes:
    print(f"Yo soy {encapuchado}")
```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Accede a la sección **Archivos del curso** del presente módulo para replicar la práctica.

Tecmilenio no guarda relación alguna con las marcas mencionadas como ejemplo. Las marcas son propiedad de sus titulares conforme a la legislación aplicable, se utilizan con fines académicos y didácticos, por lo que no existen fines de lucro, relación publicitaria o de patrocinio.

Cierre

Las estructuras dentro de los lenguajes de programación son parecidos, es cierto que las pocas diferencias pueden afectar por completo el modo de codificar en cada uno, y es que Python es bastante versátil en ese sentido, eso hace que las líneas de código se puedan simplificar, si es que se domina la estructura completa del lenguaje, obtener elementos, eliminar elementos y crear métodos o funciones que optimicen todo un código para hacerlo más sencillo en su lectura, haciendo más fácil el concepto de no repetir líneas de código y generar nuevas funciones a este.

Las funciones pueden recibir argumentos, procesarlos y devolver valores. En ocasiones, es muy útil definirle varios parámetros e incluso es posible invocar a otras funciones dentro una estructura superior. En Python, es una buena práctica encapsular el código dentro de una función principal, de esa forma se garantiza la limpieza en la sintaxis y se evitan posibles errores durante el momento de la ejecución.

Referencias bibliográficas

- Matthes, E. (2019). *Python Crash Course: A Hands-On, Project-Based Introduction to Programming* (2a ed.). Estados Unidos: No Starch Press.
- Python Software Foundation. (s.f.). *5. Estructura de datos*. Recuperado de https://docs.python.org/es/3/tutorial/datastructures.html
- Silva. M. (2019). *Don't repeat yourself.* Recuperado de https://medium.com/code-thoughts/dont-repeat-yourself-caa413910753

Para saber más

Los siguientes enlaces son externos a la Universidad Tecmilenio, al acceder a ellos considera que debes apegarte a sus términos y condiciones.

Lecturas:

- Para conocer más acerca de **estructuras de datos y funciones**, te sugerimos leer lo siguiente:
- Díaz, D. (2021). *Guía para principiantes de la Programación Orienta a Objetos (POO) en Python.* Recuperado de https://kinsta.com/es/blog/programacion-orientada-objetos-python/

Videos:

- Para conocer más acerca de **estructuras de datos y funciones**, te sugerimos revisar lo siguiente:
- Commit That Line! (2020, 30 de septiembre). *Listas, Tuples, Sets, Strings y Diccionarios en PYTHON* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=CCUNuqqn7PQ
- Tecnología Binaria; (2020, 2 de junio). *CÓMO Funciona el BUCLE WHILE en PYTHON | Curso de Python Básico #11* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=J5XiRGnRLJc

Checkpoints

Asegúrate de:

- Conocer las estructuras e identificar las diferencias entre (listas, diccionarios y conjuntos).
- Crear funciones básicas que agilicen y optimicen el código de tal manera que no sea repetitivo en todo el desarrollo.
- Reconocer las diferencias del ciclo **while** y **for** y aplicarlas según corresponda tu necesidad de desarrollo.

Requerimientos Técnicos

Python es capaz de ser utilizado para cualquier plataforma o sistema operativo existente.

En el caso de Windows se sugiere contar con:

Hardware

- Se recomienda tener con al menos 500 MB de almacenamiento disponible.
- Procesador de por lo menos 1.6 Ghz.
- 1 GB de RAM.

Plataformas

- OS X El Capitan (10.11+).
- Windows 10, 11 (32-bit and 64-bit).
- Linux (Debian): Ubuntu Desktop 16.04, Debian 9.
- Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 7, Fedora 34.

Prework

- Estudiar el tema **Estructura de datos y funciones**.
- Lee los siguientes artículos:

Los siguientes enlaces son externos a la Universidad Tecmilenio, al acceder a ellos considera que debes apegarte a sus términos y condiciones.

- Microsoft. (s.f.). *Uso de bucles "while" y "for" en Python*. Recuperado de https://docs.microsoft.com/es-es/learn/modules/python-loops/
- Microsoft. (s.f.). *Introducción a las listas en Python*. Recuperado de https://docs.microsoft.com/es-es/learn/modules/intro-python-lists/
- Microsoft. (s.f.). *Administración de datos con diccionarios de Python*. Recuperado de https://docs.microsoft.com/es-es/learn/modules/python-dictionaries/
- Microsoft. (s.f.). *Funciones de Python*. Recuperado de https://docs.microsoft.com/es-es/learn/modules/functions-python/

Tecmilenio no guarda relación alguna con las marcas mencionadas como ejemplo. Las marcas son propiedad de sus titulares conforme a la legislación aplicable, se utilizan con fines académicos y didácticos, por lo que no existen fines de lucro, relación publicitaria o de patrocinio.

La obra presentada es propiedad de ENSEÑANZA E INVESTIGACIÓN SUPERIOR A.C. (UNIVERSIDAD TECMILENIO), protegida por la Ley Federal de Derecho de Autor; la alteración o deformación de una obra, así como su reproducción, exhibición o ejecución pública sin el consentimiento de su autor y titular de los derechos correspondientes es constitutivo de un delito tipificado en la Ley Federal de Derechos de Autor, así como en las Leyes Internacionales de Derecho de Autor.

El uso de imágenes, fragmentos de videos, fragmentos de eventos culturales, programas y demás material que sea objeto de protección de los derechos de autor, es exclusivamente para fines educativos e informativos, y cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por UNIVERSIDAD TECMILENIO.

Queda prohibido copiar, reproducir, distribuir, publicar, transmitir, difundir, o en cualquier modo explotar cualquier parte de esta obra sin la autorización previa por escrito de UNIVERSIDAD TECMILENIO. Sin embargo, usted podrá bajar material a su computadora personal para uso exclusivamente personal o educacional y no comercial limitado a una copia por página. No se podrá remover o alterar de la copia ninguna leyenda de Derechos de Autor o la que manifieste la autoría del material.