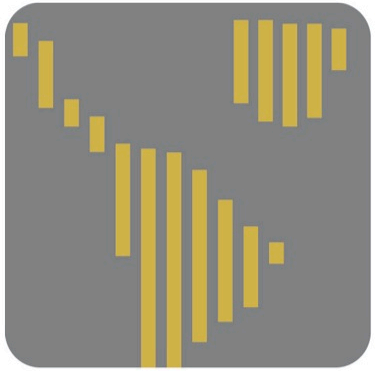


PR0MiDAT

IBEROAMERICANO

Programa Iberoamericano de Formación en Minería de Datos

www.promidat.com / info@promidat.com / (506) 4030-1205 / 4030-1114

PRØMiDAT
IBEROAMERICANO
Programa Iberoamericano de Formación en Minería de Datos

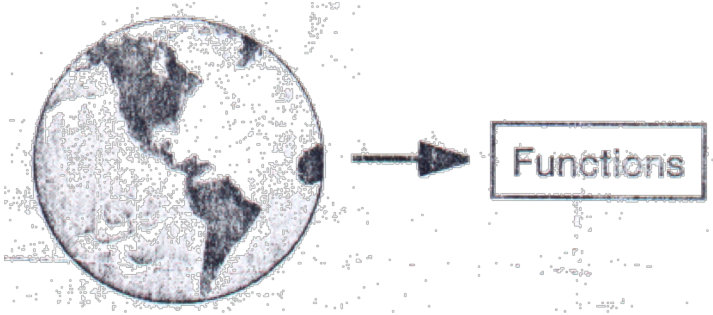
Machine Learning en Python I

Programación en Python

Definiciones
en
Orientación a Objetos

Paradigma Funcional

- Se basa en la idea de que el Dominio de una Aplicación y los Requerimientos se pueden ***modelar, programar e implementar*** basado en descomposición en ***funciones*** y sub-funciones.



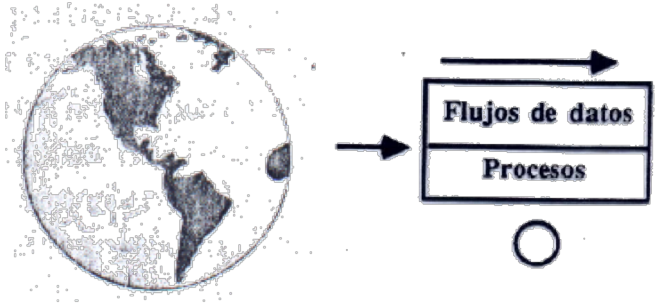
Paradigma Lógico

- Se basan en la idea de que el Dominio de una Aplicación y los Requerimientos se pueden ***modelar, programar e implementar*** basado en el uso directo del Lenguaje de la Lógica de Primer Orden

$$p \rightarrow q$$
$$\underline{q \rightarrow r}$$
$$\therefore p \rightarrow r$$

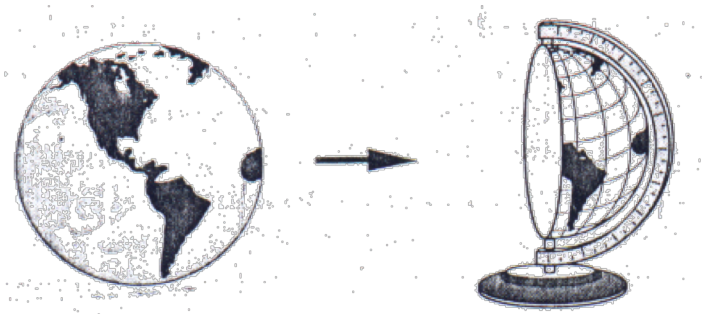
Paradigma Estructurado

- Se basan en la idea de que el Dominio de una Aplicación y los Requerimientos se pueden ***modelar, programar e implementar*** basado en:
  - PE = Procedimientos + Estructuras de Datos + Diccionario de Datos.



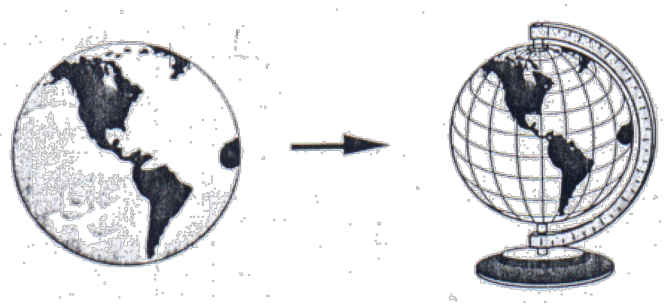
Paradigma Basado en Objetos

- Se basan en la idea de que el Dominio de una Aplicación y los Requerimientos se pueden ***modelar, programar e implementar*** basado en: BO = Objetos + Atributos + TDAs



Paradigma Orientación a Objetos

- Se basan en la idea de que el Dominio de una Aplicación y los Requerimientos se pueden ***modelar, programar e implementar*** basado:
  - OO = Objetos + Métodos + Asociaciones + Herencia + Polimorfismo



Objeto [Booch]

⇨*Un objeto es una entidad tangible que exhibe algunas conductas bien definidas.*

⇨*Un objeto tiene estado, conducta e identidad; la estructura y la conducta de objetos similares se definen en clases comunes; los términos instancia y objeto son intercambiables.*

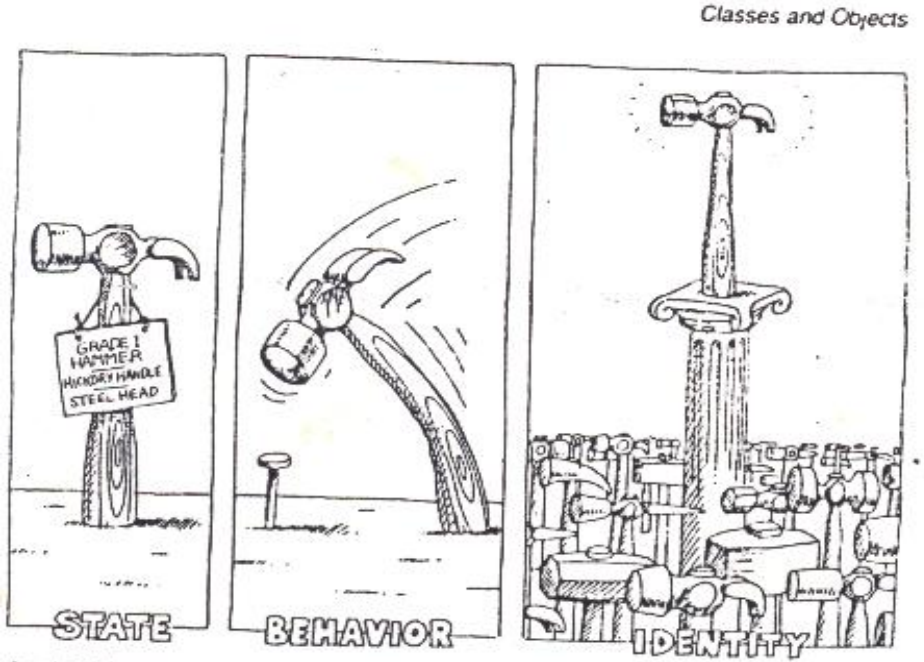
Objeto [Wegner]

- *Un objeto enmarca el estado de la computación en forma encapsulada. Cada objeto tiene una interfaz de métodos que controlan el acceso al estado encapsulado. Los métodos determinan la conducta del objeto.*

Objeto [Lécluse]

- *Una clase C es un triplete C = (i,v,m) donde i es un identificador, v es un conjunto de atributos (que puede ser vacío) y m es un conjunto de métodos (que puede ser vacío).*

Clase vrs Objeto



An object has state, exhibits some well-defined behavior, and has a unique identity.

Clase [Booch]

- *Una clase es un conjunto de objetos que comparten una estructura común y una conducta común.*
- *Es decir,* los objetos similares son agrupados en clases, las cuales reúnen los atributos y operaciones comunes a todas sus instancias.

Clase vrs Objeto (Instancia)

- Algunos autores usan *instancia, instancia de clase* o *entidad,* para referirse a un objeto. En todas las definiciones anteriores se menciona que un objeto tiene un *estado,* una *conducta*, tiene *operaciones* o *servicios* y que de alguna manera están ubicados dentro de *clases.*

Ejemplo de *clase* en Python

```
# Clase Punto
class Punto:
    def __init__(self, x = 0, y = 0): # Constructor
        self.x = x
        self.y = y
    def cambiar(self, x, y):
        self.x = x
        self.y = y
    def inicializar(self):
        self.cambiar(0, 0)
    def calcular_distancia(self, otro_Punto):
        return math.sqrt(
                (self.x - otro_Punto.x)**2 +
                (self.y - otro_Punto.y)**2)
```

Ejemplo de ***instancia*** en Python

```
# ¿Cómo usarla?
Punto1 = Punto(2,3)
Punto2 = Punto(-2,9)
print(Punto1.x)
print(Punto1.y)
print(Punto2.x)
print(Punto2.y)
```

Estado de un objeto [Booch]

- *El estado de un objeto abarca todas las propiedades (usualmente estáticas) del objetos más los valores.*
- Se puede decir entonces, en términos muy orientados a la programación, que el estado de un objeto lo determina el valor (en un tiempo dado) de atributos de la clase.

Estado de un objeto [Booch]

```
# ¿Cómo usarla?
Punto1 = Punto(2,3)
Punto2 = Punto(-2,9)
print(Punto1.x)
print(Punto1.y)
print(Punto2.x)
print(Punto2.y)

Punto1.inicializar()
Punto2.cambiar(5,0)
```

¿Qué es un método? [Snyder]

- *Un método es un procedimiento o función que ejecuta los servicios. Típicamente un objeto tiene un método para cada operación que soporta. Los métodos son frecuentemente definidos en la implementación del objeto permitiendo que las variables de estado sean leídas y moficadas.*

Ejemplo de *clase* en Python

```
# Clase Punto
class Punto:
    def __init__(self, x = 0, y = 0): # Constructor
        self.x = x
        self.y = y
    def cambiar(self, x, y):
        self.x = x
        self.y = y
    def inicializar(self):
        self.cambiar(0, 0)
    def calcular_distancia(self, otro_Punto):
        return math.sqrt(
                (self.x - otro_Punto.x)**2 +
                (self.y - otro_Punto.y)**2)
```

Conducta de un objeto [Booch]

- *La conducta es: cómo un objeto actúa y reacciona, en términos de los cambios de su estado y el paso de mensajes*
- Se puede decir que la conducta de un objeto está determinada por sus *métodos* (funciones miembro, operaciones o servicios).

Conducta de un objeto [Booch]

```
print(Punto2.calcular_distancia(Punto1))
Punto1.cambiar(3,4)
print(Punto1.calcular_distancia(Punto2))
print(Punto1.calcular_distancia(Punto1))
```

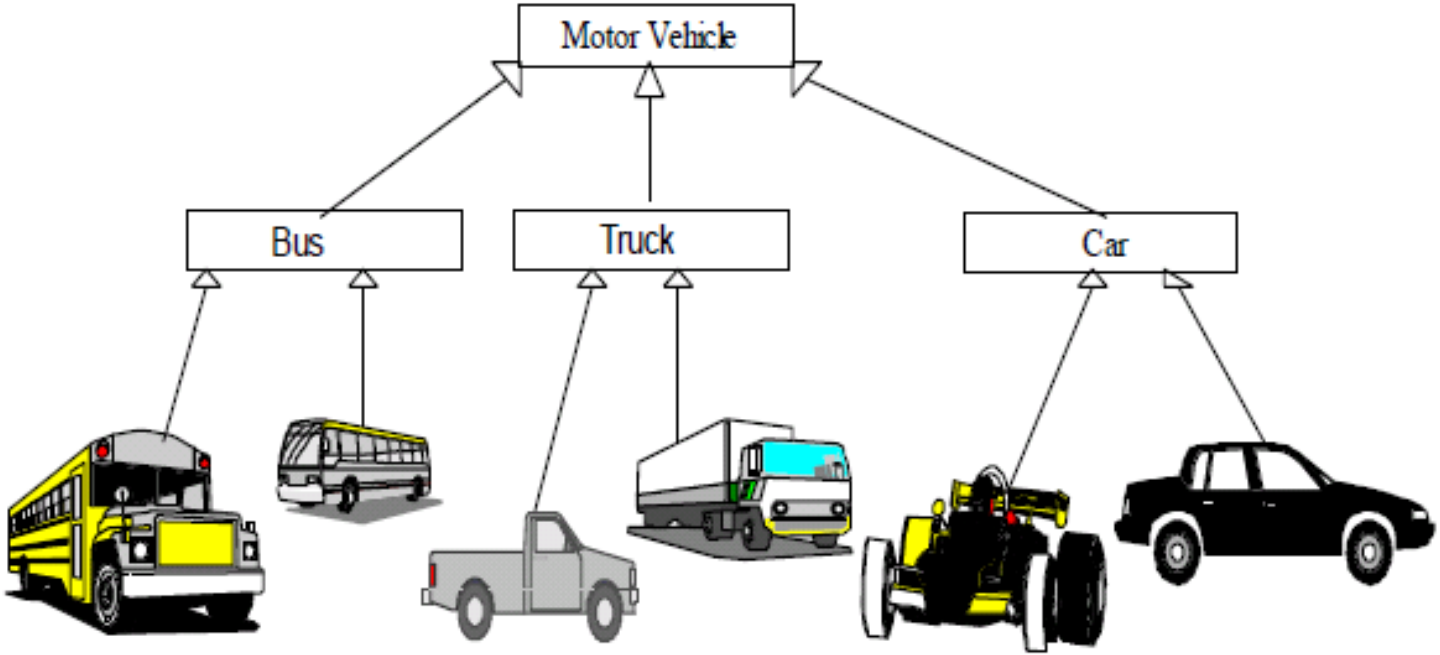
Conducta de un objeto [Booch]

```
# Ejemplo de conducta de clase
class Perro():
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def sentarse(self):
        print(self.nombre.title() + " está ahora sentado.")
    def levantarse(self):
        print(self.nombre.title() + " se levantó!")


mi_Perro = Perro('Puppy', 14)
su_Perro = Perro('Negro', 3)

print("El nombre de mi perro es " + mi_Perro.nombre.title() + ".")
print("Mi perro tiene " + str(mi_Perro.edad) + " de edad.")
mi_Perro.sentarse()

print("El nombre de su perro es " + su_Perro.nombre.title() + ".")
print("Su perro tiene " + str(su_Perro.edad) + " de edad.")
su_Perro.sentarse()

mi_Perro.levantarse()
su_Perro.levantarse()
```

Herencia de Clases

- La herencia es uno de los mecanismos fundamentales en los lenguajes de programación orientados a objetos, además, es el principal mecanismo mediante el cual se permite la reutilización de código, algunos autores incluso dicen que un lenguaje que no soporte la herencia no es orientado a objetos, sino que *es basado en objetos*.

Herencia de Clase



Herencia [Wegner]

- *La herencia es un mecanismo para compartir el código o la conducta comunes de una colección de clases. Las propiedades compartidas se ubican en superclases y éstas se reutilizan en la definición de subclases. Los programadores pueden especificar cambios incrementales en la conducta de las subclases sin modificar la clase ya especificada. Las subclases heredan el código de la superclase y ellas pueden agregar nuevos métodos y atributos.*

Herencia [Snyder]

- *La implementación de la herencia es un mecanismo para crear la implementación de clases en forma incremental; la implementación de una clase se define en términos de la implementación de otra clase. La nueva implementación puede ser extendida agregando datos (atributos) a la nueva representación de la clase, agregando nuevas operaciones (métodos), y cambiando o extendiendo la definición de operaciones ya existentes.*

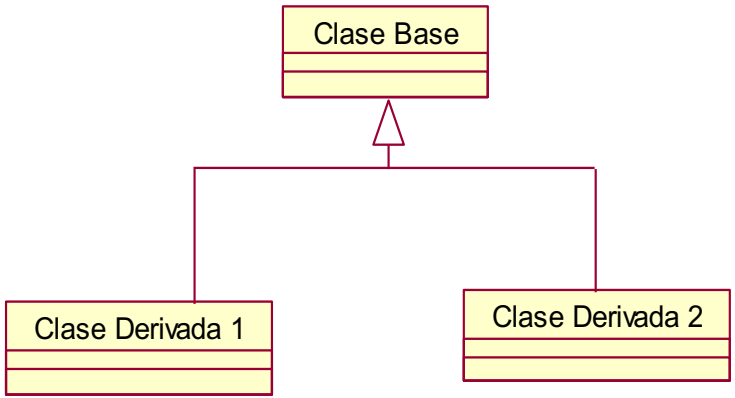
Sinónimos de Herencia

- Para el término herencia otros autores utilizan *jerarquía de clases, jerarquía de tipos,* o *jerarquía de interfaces*; para el término superclase también se utilizan los términos *clase base* o *super tipo* y para el término subclase se utiliza comúnmente *clase derivada* o *subtipo.*

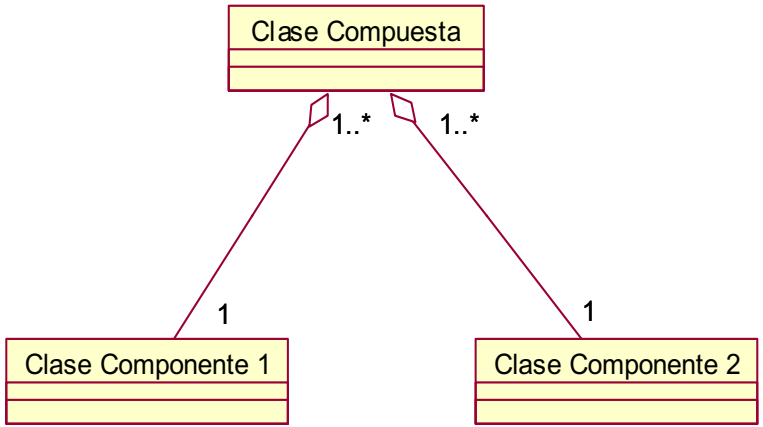
Herencia en Python

```python
# Clase Base
class Carro():
    def __init__(self, fabricante, modelo, anno):
        self.fabricante = fabricante
        self.modelo = modelo
        self.anno = anno
        self.kilometraje = 0
    def obtener_nombre_completo(self):
        nombre_completo = self.fabricante + ' ' + str(self.anno) + ' ' + self.modelo
        return nombre_completo.title()
    def lee_kilometraje(self):
        print("Este carro tiene " + str(self.kilometraje) + " kilometros recorridos.")
    def modifica_kilometraje(self, kilometros):
        if kilometros >= self.kilometraje:
            self.kilometraje = kilometros
        else:
            print("Usted no puede modificar para atrás el kilometraje!")
    def incrementa_kilometraje(self, kilometros):
        self.kilometraje += kilometros

# Clase derivada o clase hija
class Carro_Electrivo(Carro):
    def __init__(self, fabricante, modelo, anno):
        super().__init__(fabricante, modelo, anno)
        self.tamanno_bateria = 70
    def describe_bateria(self):
        print("Este Carro tiene una bateria tamaño " + str(self.tamanno_bateria) + "-kWh.")
```

Notación gráfica de la Herencia en UML



Relaciones de ***Composición*** (Com-Com) en UML



Com-Com en Python

```python
# Clase Base
class Carro():
    def __init__(self, fabricante, modelo, anno):
        self.fabricante = fabricante
        self.modelo = modelo
        self.anno = anno
        self.kilometraje = 0
    def obtener_nombre_completo(self):
        nombre_completo = self.fabricante + ' ' + str(self.anno) + ' ' + self.modelo
        return nombre_completo.title()
    def lee_kilometraje(self):
        print("Este carro tiene " + str(self.kilometraje) + " kilometros recorridos.")
    def modifica_kilometraje(self, kilometros):
        if kilometros >= self.kilometraje:
            self.kilometraje = kilometros
        else:
            print("Usted no puede modificar para atrás el kilometraje!")
    def incrementa_kilometraje(self, kilometros):
        self.kilometraje += kilometros

# Clase derivada o clase hija
class Carro_Electrico(Carro):
    def __init__(self, fabricante, modelo, anno, capacidad_bateria, marca_bateria):
        super().__init__(fabricante, modelo, anno)
        self.bateria = Bateria(capacidad_bateria, marca_bateria)

mi_tesla = Carro_Electrico('tesla', 'modelo S', 2018, 65, "Hitachi")
print(mi_tesla.obtener_nombre_completo())
mi_tesla.bateria.describe()
```

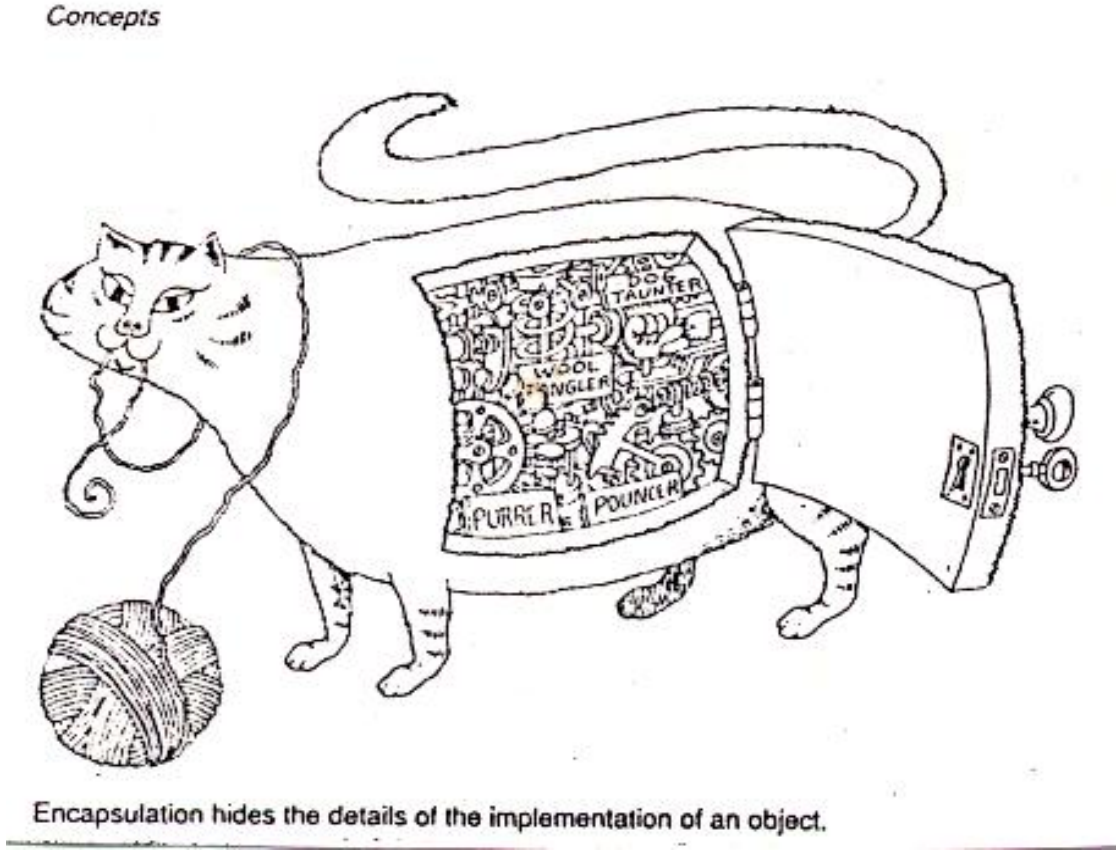
Encapsulación

La encapsulación es uno de los conceptos fundamentales de la Orientación a Objetos, en la que es un mecanismo que permite a los programadores utilizar clases sin conocer los detalles de implementación de éstas, permitiendo que futuras mejoras o cambios en la clase no impliquen cambios en los demás módulos que utilizan tal instancia de esta clase. Además, es el concepto que marca una diferencia sustancial entre un *Tipo de Dato Abstracto* (TDA) y un objeto.

Encapsulación [Booch]

- *La encapsulación es un proceso mediante el cual se ocultan todos los detalles de un objeto que no contribuyen a sus características escenciales.*

Encapsulación



Encapsulation hides the details of the implementation of an object.

Encapsulación en Python

(Atributos y Métodos Privados)

```
class Celsius:
    def __init__(self, temperatura = 0):
        self.__temperatura = temperatura
    def convierte_fahrenheit(self):
        return (self.temperatura * 1.8) + 32
    @property
    def temperatura(self):
        print("Obteniendo el valor")
        return self.__temperatura
    @temperatura.setter
    def temperatura(self, nuevo_valor):
        if nuevo_valor < -273:
            raise ValueError("Una temperatura por debajo de -273 no es posible")
        print("Retornando el valor")
        self.__temperatura = nuevo_valor
```

PRØMiDAT
IBEROAMERICANO

Programa Iberoamericano de
Formación en Minería de Datos

www.promidat.com

info@promidat.com

(506) 4030-1205 / 4030-1114