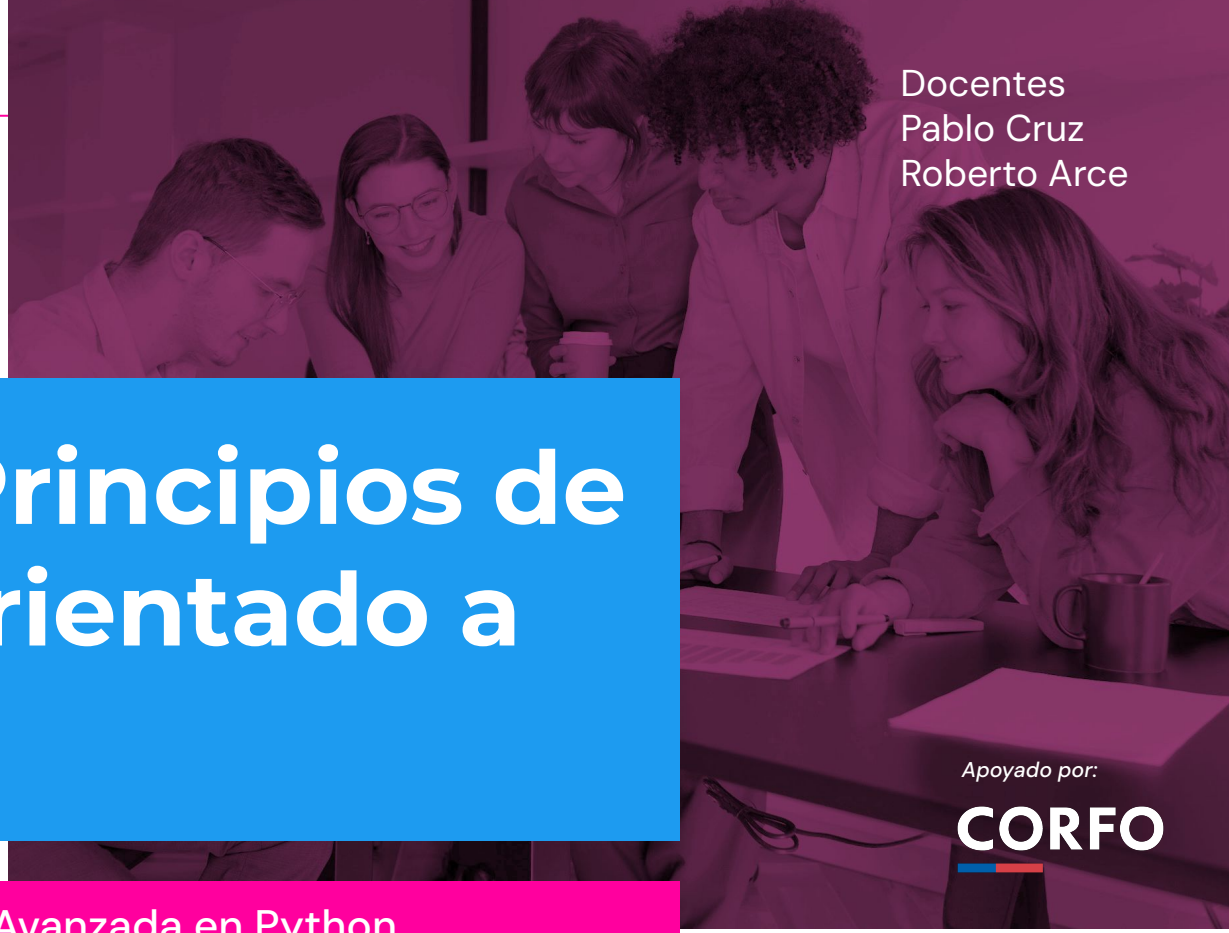
TALENTOFUTURO



Docentes
Pablo Cruz
Roberto Arce

Clase 4: Principios de Diseño Orientado a Objetos

Módulo 2: Programación Avanzada en Python.

Apoyado por:
CORFO

Clase de hoy

01

Clases, objetos, métodos

Clases, objetos y métodos.

02

Principios del diseño

Encapsulamiento, herencia y polimorfismo.

03

Representación de clases

Diagramas UML.

4A

Clases, objetos, métodos y principios de diseño

Bloque A

Qué veremos en Bloque A

- Clases y objetos.
- Métodos.
- Generalización versus especificación.

Orientación a Objetos

Un poco de historia...

- La orientación a objetos es un paradigma de implementación de programas que emerge en los años ’80.
- El paradigma es adoptado inicialmente por lenguajes como Smalltalk 80 y C++.
- La orientación a objetos representa la continuación de la evolución de los lenguajes de programación hacia un enfoque “más alejado de la máquina, más cercano al dominio o problema”.
- En la actualidad, Python es un lenguaje que puede ser considerado multiparadigma y como tal la orientación a objetos es uno de los paradigmas adoptados.

Orientación a Objetos

¿Qué es la orientación a objetos?

- La orientación a objetos es un paradigma, es decir, un enfoque para enfrentar la complejidad de sistemas usando el principio de descomposición por objetos.
- En el paradigma funcional, la descomposición es por funciones o “lo que se hace o se espera hacer en el dominio”.
- En el paradigma de la orientación a objetos, la descomposición es por elementos que tienen características y comportamiento definido.
- En el centro de la orientación a objetos está el modelo de objetos que típicamente representaremos como un diagrama de clases u objetos.

Orientación a Objetos

Clases

- La “clase” es un concepto clave en la orientación a objetos: representa una abstracción, es decir, agrupa aspectos similares entre objetos, situaciones o procesos de la vida real.
- Esta abstracción define una frontera conceptual que nos permite determinar y distinguir distintas clases.
- No es correcto asumir que una clase representa solo objetos de la vida real: las abstracciones pueden ser de tipo entidades (típicamente objetos de la vida real), acciones, máquinas, casuales, entre otras.
- Por lo tanto, no limitaremos nuestro modelo de objetos a solo objetos de la vida real.

Orientación a Objetos

Objetos

- Una clase representa la esencia de un objeto: diremos que una clase representa a un conjunto de objetos que pueden ser derivados desde “el molde” que provee la clase.

- En los lenguajes de programación típicamente se entiende que un objeto es una instancia concreta de una clase.

- Sin embargo, tenemos dos diferencias esenciales y que van más allá de simples apreciaciones:
    - Las clases son estáticas: es decir, la existencia de estas se define antes de la ejecución de un programa.
    - Los objetos son dinámicos: es decir, son creados y destruidos en la ejecución del programa.

Orientación a Objetos

Atributos y Métodos

- En orientación a objetos diremos que las variables que son comprendidas por el objeto (o clase) son los atributos de este.
- Los atributos definen las características y el estado del objeto.
- Por otro lado, el comportamiento del objeto estará expresado por medio de funciones.
- A las funciones que son comprendidas por el objeto (o clase) les llamaremos “métodos”.
- Esta distinción es importante porque un principio importante en la orientación a objetos es que el estado del objeto se determina y modifica solamente por medio de los métodos que este expone.

Orientación a Objetos

Clases y Objetos en Python

- Definir una clase:

```
class Persona:
        nombre = “Juan Pérez”

        def get_nombre(self):
        return self.nombre
```

- Creación de un objeto (instancia de la clase):

```
x = Persona()
print(x.get_nombre())
```

Orientación a Objetos

Clases y Objetos en Python

- Definir una clase con constructor:

```
class Persona:
        def __init__(self, nombre):
        self.nombre = nombre

        def get_nombre():
        return self.nombre
```

- Creación de un objeto (instancia de la clase):

```
x = Persona("Juan Pérez")
print(x.get_nombre())
```

Orientación a Objetos

Clases y Objetos en Python

- Como vemos, los atributos y métodos se declaran en forma muy similar a como estamos acostumbrados, pero con algunas diferencias.
- La primera diferencia es el uso de la palabra “self”.
- “self” se utiliza para indicar los métodos y atributos que son parte de “la clase” y no una clase superior (concepto de generalización).
- La segunda diferencia es que el método debe ser diferenciado respecto de los nombres de las variables o atributos.

Orientación a Objetos

Generalización versus Especificación

- En ocasiones, algunas abstracciones tienden a agrupar características comunes a muchas otras abstracciones.
- A las primeras las llamaremos abstracciones generales (o más generales) y a las segundas las llamaremos abstracciones específicas (o más específicas).
- Ejemplo:

```
class Student:
    ……

class UndergraduateStudent(Student):
    ……
```

Trabajo grupal – Bloque A

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		

G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal – Ejercicio #1

Creación de objetos complejos

- Cree las clases Waypoint y Trackpoint.
- La clase Waypoint contiene un nombre; la clase Trackpoint contiene una fecha de registro.
- Ambas clases son posiciones geográficas de tipo Position.
- La clase Position requiere en su inicialización una latitud, una longitud y una altitud.

Trabajo grupal – Ejercicio #2

Múltiples representaciones

- Cree dos representaciones para los objetos de tipo posición:
    - Una representación de atributos separados por coma (string).
    - Una representación de atributos como diccionario.

Trabajo grupal – Ejercicio #3

Distancia geodésica

- Importe la biblioteca “geopy” para hacer uso de la función de distancia geodésica `(import geopy.distance)`.
- Cree una clase “helper” llamada Distance que se inicialice con dos posiciones: una posición de origen y una posición de destino.
- Cree un método llamado “km(self)” que retorne la distancia geodésica en kilómetros.

```
return geopy.distance.geodesic(
            ((self.source.__dict__())["latitude"],
             (self.source.__dict__()["longitude"])),
            ((self.destination.__dict__())["latitude"],
             (self.destination.__dict__()["longitude"]))
        ).km
```

Break!

4B

Representación de Clases

Bloque B

Qué veremos en Bloque B

- Principios de diseño en la orientación a objetos.
- Introducción a UML.
- Representación de clases en UML.

Principios de Diseño

Principios del diseño orientado a objetos

- Los principios son definiciones generales comúnmente aceptadas y que aplican a un contexto particular.

- En nuestro caso, los principios de diseño nos guían en la creación de los modelos de objetos que finalmente serán implementados en el software.

- Revisaremos tres principios:
  - Encapsulamiento.
  - Generalización (comúnmente llamada “herencia”).
  - Polimorfismo.

Principios de Diseño

Encapsulamiento

- El principio de encapsulamiento nos dice que debemos ocultar la implementación de un objeto.
- Como consecuencia, el principio establece que la única forma en que podemos acceder a revisar o modificar el estado (es decir, los valores de las variables) de un objeto es por medio de las interfaces establecidas.
- Las interfaces establecidas se definen por medio de los métodos.
- En el caso de ejemplo, el método get_nombre() nos permite obtener el valor actual de la variable nombre sin recurrir al acceso directo a la variable nombre.

Principios de Diseño

Generalización

- Este principio reconoce que las abstracciones no son de un solo nivel.
- Tenemos abstracciones más específicas y abstracciones más generales.
- Por ejemplo, la abstracción “Estudiante” es más específica que la abstracción “Persona”.
- Asimismo, las abstracciones (por ejemplo) “Curso de diplomado” y “Curso de pregrado” son más específicas que la abstracción “Curso”.
- En la generalización esperamos que la clase más general contenga los elementos (atributos y métodos) que son comunes a todas las abstracciones más específicas.

Principios de Diseño

Generalización

- En Python expresamos la generalización versus especificación por medio del uso de sub clases.
- Si tenemos la clase Persona, la clase Estudiante se definirá como:
  class Estudiante(Persona):...
- En el ejemplo, Estudiante es una clase más específica que Persona.
- La generalización muchas veces se usa como integridad y consistencia conceptual. En otras ocasiones se usa con el objetivo operacional de heredar elementos.
- Si hay herencia de elementos diremos que la especificación es también herencia.

Principios de Diseño

Polimorfismo

- El principio de polimorfismo significa que un elemento puede tener distintas formas.
- Aunque el principio depende en su implementación del lenguaje usado, típicamente observamos el principio en los métodos.
- Los métodos que usan polimorfismo típicamente exponen una interfaz igual o similar, pero se construyen de formas distintas.
- Por ejemplo, en Python observamos el método len() que se encuentra presente en muchos objetos, pero cada uno de estos lo implementa de manera distinta (ejemplo: len(string) versus len(lista)).

Unified Modeling Language

UML

- El lenguaje UML (Unified Modeling Language) es un estándar OMG (Object Management Group) que define una sintaxis y semántica para la representación de modelos de objetos.

- En UML tenemos distintos diagramas, pero en esta parte del curso nos preocuparemos de los diagramas de clases que son usados para representar clases.

- Veremos como las clases se relacionan con generalización y asociaciones simples y de tipo composición.

Unified Modeling Language

Diagrama de Clases

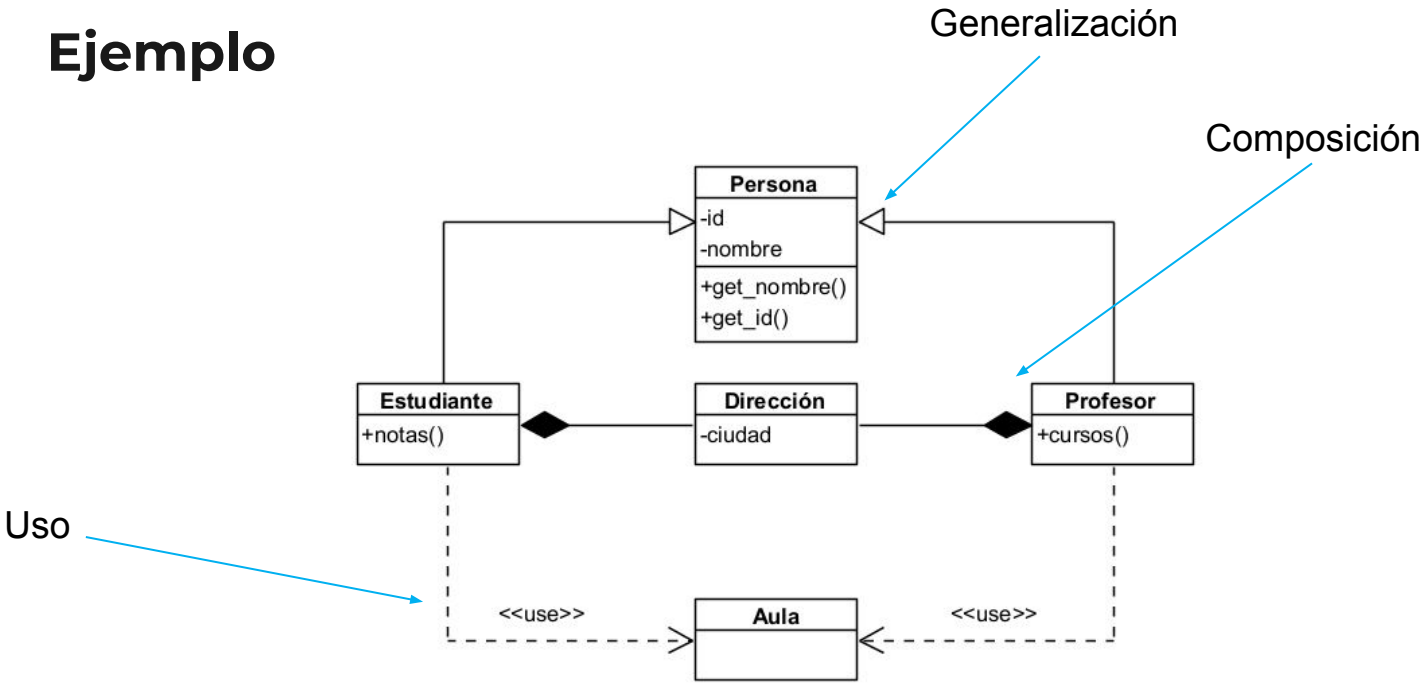
- Un diagrama de clases presenta las clases como cajas con compartimentos que permiten expresar el nombre de la clase, los atributos de la clase, y los métodos de la clase.

- El diagrama de clases permite expresar una o más clases.

- Las clases son relacionadas por medio de las siguientes relaciones:
  - Generalización: una clase es más general que otra.
  - Composición: una clase se compone de otra (u otras) clases.
  - Uso: una clase hace uso genérico de otra clase.

Diagrama de Clases

Ejemplo



Trabajo grupal – Bloque B

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		

G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal – Ejercicio #1

Instalación de Visual Paradigm Community Edition

- Baje el software Visual Paradigm en su versión Community Edition en el siguiente enlace: https://www.visual-paradigm.com/download/community.jsp
- Instale el software en su computador.
- Inicie el software, asigne un directorio para sus modelos, y cree un diagrama de clases (sin asignar clases).

Trabajo grupal – Ejercicio #2

Modelo de clases

- Construya un modelo de clases para el caso del ejercicio anterior (Bloque A).
- Presente y discuta el modelo.

TALENTOFUTURO

¿Preguntas?

¡Hemos llegado al final de la clase!