Universitat Autònoma de Barcelona

2020/2021

Programación Orientada a los Objetos

Código: 104389
Créditos ECTS: 6

Titulación	Tipo	Curso	Semestre
2503740 Matemática Computacional y Analítica de Datos	FB	1	2

La metodología docente y la evaluación propuestas en la guía pueden experimentar alguna modificación en función de las restricciones a la presencialidad que impongan las autoridades sanitarias.

Contacto

Nombre: Joan Serrat Gual
Correo electrónico: Joan.Serrat@uab.cat

Uso de idiomas

Lengua vehicular mayoritaria: catalán (cat)
Algún grupo íntegramente en inglés: No
Algún grupo íntegramente en catalán: Sí
Algún grupo íntegramente en español: No

Equipo docente

Lluis Gomez Bigorda

Prerequisitos

No hay prerrequisitos oficiales, pero es necesario saber programar en Python.

Objetivos y contextualización

Una de las metodologías más extendidas de diseño y programación es la orientada a objetos, según la cual un software se organiza en clases que contienen métodos (procedimientos) y atributos (datos). Las instancias u objetos de estas clases se envían mensajes unas a otras (hacen "llamadas" a métodos de otros objetos) consiguiendo así la funcionalidad deseada. Aparte del concepto de clase, tres otros elementos fundamentales son la herencia, la composición y la separación de las partes pública y privada de las clases. Estos elementos son pues necesarios para la programación OO, pero no suficientes: hay que saber cómo utilizarlos para llegar a soluciones que sean fácilmente extensibles en el futuro, ya que una ley del software es que los cambios son inevitables. Existen una serie de principios de diseño o heurísticas que nos dicen cómo emplear estos elementos con el fin de conseguirlo. Y sobre ellos todavía, se construyen los patrones de diseño, que son soluciones a nivel de diseño OO a problemas recurrentes, que aparecen a menudo en resolver problemas aparentemente independientes. En esta asignatura se introducen y se practican los conceptos OO así como los principios y patrones de diseño.

El aprendizaje gira en torno a un proyecto --- la práctica --- en que es necesario aplicar todos los temas explicados para diseñar e implementar una aplicación de software a partir de un enunciado que hace el papel de documento de requerimientos. Por el contexto de la asignatura, el proyecto cae en el campo del aprendizaje computacional (machine learning). Los lenguajes de programación utilizados en la asignatura serán Python principalmente por que es en el que se hará el proyecto, pero también Java que utilizaremos en algunos ejercicios y soporta mejor los conceptos de orientación a objeto. Así,podremos comparar cómo uni otro los implementa y en qué grado.

Competencias

-

- Aplicar conocimientos básicos sobre la estructura, el uso y la programación de ordenadores, sistemas operativos y programas informáticos para solucionar problemas de distintos ámbitos.
- Diseñar, desarrollar y evaluar soluciones algorítmicas eficientes para problemas computacionales de acuerdo con los requisitos establecidos.
- Evaluar de manera crítica y con criterios de calidad el trabajo realizado.
- Que los estudiantes hayan demostrado poseer y comprender conocimientos en un área de estudio que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados, incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio.
- Que los estudiantes puedan transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado.
- Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.
- Trabajar cooperativamente en un contexto multidisciplinar asumiendo y respetando el rol de los diferentes miembros del equipo.
- Utilizar eficazmente bibliografía y recursos electrónicos para obtener información.

Resultados de aprendizaje

1. Comprender los principios básicos de la lógica de los computadores.
2. Conocer los conceptos básicos de la estructura y la programación de los computadores.
3. Describir el funcionamiento básico de los sistemas de cómputo.
4. Evaluar de manera crítica y con criterios de calidad el trabajo realizado.
5. Evaluar y analizar la complejidad computacional de las soluciones algorítmicas para poder desarrollar e implementar aquella que garantice el mejor rendimiento.
6. Que los estudiantes hayan demostrado poseer y comprender conocimientos en un área de estudio que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados, incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio.
7. Que los estudiantes puedan transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado.
8. Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.
9. Reconocer e identificar los métodos, sistemas y tecnologías propios de la computación.
10. Seleccionar y utilizar las estructuras algorítmicas y de representación de los datos apropiadas para la resolución de un problema.
11. Trabajar cooperativamente en un contexto multidisciplinar asumiendo y respetando el rol de los diferentes miembros del equipo.
12. Utilizar eficazmente bibliografía y recursos electrónicos para obtener información.
13. Utilizar los sistemas operativos y programas de uso común en distintos campos.
14. Verificar y asegurar el funcionamiento correcto de una solución algorítmica de acuerdo con los requisitos del problema a resolver.

Contenido

1. Conceptos de orientación a objeto: encapsulamiento, herencia, composición, interfaz. Diagramas de clase UML
2. Programación orientada a objeto en Python3 y Java
3. Asignación de responsabilidades: patrones GRASP
4. Principios generales de diseño orientado a objeto
5. Patrones de diseño: creacionales, estructurales, de comportamiento
6. Estilo de codificación, aserciones y mensajes de logging

Metodología

Tal como hemos dicho a los objetivos, la asignatura sigue una metodología de aprendizaje basado en el desarrollo de un proyecto. Su implementación concreta en cuanto al uso de las horas de clase es la siguiente:

- Clases de teoría. En ellas el profesor introduce los contenidos teóricos y da referencias útiles (libros, artículos, páginas web) para que el estudiante después pueda continuar su aprendizaje, así como desarrollar el proyecto propuesto. Las transparencias utilizadas en estas clases pues no deben tomarse como única fuente de estudio. Hay que estudiar los libros recomendados, y los artículos, capítulos de libros de acceso abierto y otros, disponibles en la página web de la asignatura. En estas clases también resolveremos pequeños problemas, de diseño y / o programación OO en Java y Python.

- Sesiones de prácticas. Los estudiantes deberán formado a principio de curso grupos de 3 estudiantes. En estas sesiones los estudiantes vienen a 1) 'ajustar cuentas' con el profesor del trabajo hecho cada semana del proyecto y resolver dudas, y 2) a ser evaluados de manera continua, de acuerdo con el baremo que figura en el enunciado de la práctica. Por lo tanto, la práctica *no* se hace sólo durante estas sesiones sino que pide un trabajo previo.

Todos los materiales de la asignatura junto con su planificación detallada se encontrarán en el campus virtual o bien a una página web que ya se anunciará. La mayoría de estos materiales son en inglés.

Actividades

Título	Horas	ECTS	Resultados de aprendizaje
Tipo: Dirigidas			
Clases de teoria	75	3	5, 2, 3, 9, 10
clases de prácticas	75	3	7, 11, 13

Evaluación

Hay una parte de evaluación de grupo y otra individual. La nota de la primera (PR) corresponde a la práctica. La segunda (EX) del promedio de dos pruebas escritas durante el trimestre (EX1, EX2), o bien de un examen de recuperación (EXR1, EXR2).

El algoritmo para el cálculo de la nota final (FM) es el siguiente (todas las notas son sobre 10):

EX = 0.5 * (max (EX1 + EXR1), max (EX2, EXR2))
if (EX> = 4.0) and (PR> = 4.0):
FM = (EX + PR) /2.0
else:
FM = min (EX, PR)

Opcionalmente, propondremos algunos ejercicios y pequeños problemas que sumarán puntos a los exámenes EX1, EX2 y los correspondientes de recuperacion.

Si no se entrega un examen o la práctica, la nota de esta parte es cero. No hay recuperación de la práctica. La nota final será 'no evaluable' si no se ha entregado nada evaluable, ya sea examen o práctica.

Algo que queremos asegurar mediante las diferentes evaluaciones es que cada miembro del grupo realmente ha contribuido a la práctica en la misma medida que los demás. Así pues, si de resultas de las preguntas que el profesor hace durante las sesiones de proyecto o por las notas de las pruebas escritas tenemos indicios de que esto no ocurre, entonces la nota de proyecto pasará de ser de grupo a individual: evaluaremos

separadamente la parte del proyecto que haya hecho cada miembro, pudiendo además dividir el grupo a partir de ese momento.

Sin perjuicio de otras medidas disciplinarias que se estimen oportunas, y de acuerdo con la normativa académica vigente, se calificarán con un cero las irregularidades cometidas por el estudiante que puedan conducir a una variación de la calificación de un acto de evaluación. Por lo tanto, plagiar, copiar o dejar copiar la práctica o cualquier otra actividad de evaluación implicará suspender con un cero y no se podrá recuperar en el mismo curso académico. Si esta actividad tiene una nota mínima asociada, entonces la asignatura quedará suspendida.

Actividades de evaluación

Título	Peso	Horas	ECTS	Resultados de aprendizaje
Exámenes	50%	0	0	5, 2, 3, 6, 7, 8, 9, 10
Práctica	50%	0	0	4, 1, 2, 3, 7, 9, 11, 12, 13, 14

Bibliografía

Python: master the art of design patterns. D. Phillips, C. Giridhar, S. Kasampalis. Packt Publishing, 2016.

Mastering object-oriented Python. Steven F. Lott. Packt publishing, 2014.

Head first object oriented analysis and design. B.D. McLaughlin, G. Pollice, D. West. O'Reilly, 2007.

Design patterns explained simply. Alexander Shvets. https://sourcemaking.com/design-patterns-ebook.

Head first design patterns. E. Freeman, E. Freeman, K. Sierra, B. Bates. O'Reilly, 2004.

Clean code: a handbook of agile software craftmanship. R.C. Martin. Prentice Hall, 2008. O la versión en castellano, Código limpio, d'Anaya
Multimedia 2012.