26-27

GRADO EN INGENIERÍA EN INTELIGENCIA ARTIFICIAL
PRIMER CURSO

GUÍA DE ESTUDIO PÚBLICA

PROGRAMACIÓN ORIENTADA A OBJETOS (INGENIERÍA EN IA)

CÓDIGO 71031062

UNED

26-27

PROGRAMACIÓN ORIENTADA A OBJETOS (INGENIERÍA EN IA)

CÓDIGO 71031062

ÍNDICE

PRESENTACIÓN Y CONTEXTUALIZACIÓN
REQUISITOS Y/O RECOMENDACIONES PARA CURSAR LA ASIGNATURA
EQUIPO DOCENTE
HORARIO DE ATENCIÓN AL ESTUDIANTE
TUTORIZACIÓN EN CENTROS ASOCIADOS
COMPETENCIAS QUE ADQUIERE EL ESTUDIANTE
RESULTADOS DE APRENDIZAJE
CONTENIDOS
METODOLOGÍA
SISTEMA DE EVALUACIÓN
BIBLIOGRAFÍA BÁSICA
BIBLIOGRAFÍA COMPLEMENTARIA
RECURSOS DE APOYO Y WEBGRAFÍA
IGUALDAD DE GÉNERO

NOMBRE DE LA ASIGNATURA	PROGRAMACIÓN ORIENTADA A OBJETOS (INGENIERÍA EN IA)
CÓDIGO	71031062
CURSO ACADÉMICO	2026/2027
DEPARTAMENTO	LENGUAJES Y SISTEMAS INFORMÁTICOS
TÍTULO EN QUE SE IMPARTE	GRADO EN INGENIERÍA EN INTELIGENCIA ARTIFICIAL
CURSO	PRIMER CURSO
PERIODO	SEMESTRE 2
Nº ETCS	6
HORAS	150.0
IDIOMAS EN QUE SE IMPARTE	CASTELLANO

PRESENTACIÓN Y CONTEXTUALIZACIÓN

Programación Orientada a Objetos (POO) es una asignatura de Formación Básica (FB) de **6 créditos ECTS** que se imparte en el **segundo semestre del primer curso** del Grado en Ingeniería en Inteligencia Artificial de la UNED. Pertenece a la materia «Programación y Algoritmia».

La asignatura parte de los conocimientos de programación estructurada adquiridos en Fundamentos de Programación (primer semestre) e introduce el paradigma orientado a objetos con **Python 3** como lenguaje de trabajo. A lo largo del semestre los estudiantes aprenden a organizar el código en clases y objetos, a aplicar principios de diseño de calidad (SOLID) y patrones de diseño básicos, y a documentar, depurar y verificar sus programas de forma sistemática. Estas competencias son directamente aplicables al desarrollo de software para Inteligencia Artificial.

La asignatura se imparte en **modalidad no presencial** según el modelo propio de la UNED: estudio autónomo a partir de los materiales disponibles en la plataforma **Ágora**, apoyo de los tutores y tutoras en los Centros Asociados, y actividades prácticas con Google Colab y Visual Studio Code.

La asignatura de POO se apoya en los conocimientos adquiridos en la asignatura Fundamentos de Programación y proporciona los fundamentos del paradigma orientado a objetos que necesitan asignaturas posteriores como Estructuras de Datos y Algoritmos, que exige conocer el diseño de clases para implementar tipos abstractos de datos, Algoritmia para la IA, que requiere haber superado POO como requisito de entrada, e Introducción a la Ingeniería del Software, que retoma el diseño orientado a objetos desde una perspectiva de ingeniería del software más amplia.

Contextualización en el plan de estudios

Asignatura	Curso	Tipo	Relación con POO

Fundamentos de Programación	1.º, S1	FB	Requisito previo. Proporciona las bases de programación estructurada (secuencia, selección, iteración, funciones, tipos de datos). Python 3 se introduce como primer lenguaje OO en el Bloque I de POO.
Programación Orientada a Objetos asignatura actual	1.º, S2	FB	
Estructuras de Datos y Algoritmos	2.º, S1	FB	Posterior. Exige los conocimientos de FP y POO. Aplica el diseño de clases para implementar estructuras de datos (recursión, estructuras indexadas, secuenciales y jerárquicas).
Algoritmia para la IA	2.º, S2	OB	Posterior. Requiere haber superado FP, POO y EDA. Trabaja complejidad algorítmica, esquemas algorítmicos y algoritmos de búsqueda en IA.

Introducción a la Ingeniería del Software	2.º, S2	OB	Posterior. Amplía el diseño orientado a objetos desde la perspectiva de la ingeniería del software: ciclo de vida, especificación de requisitos, UML como lenguaje de modelado y pruebas del software.
Diseño de Aplicaciones Orientadas a Objetos	4.º, S2	OP	Optativa posterior. Requiere haber superado POO. Profundiza en el diseño OO: introducción a los patrones de diseño (creacionales, estructurales y de comportamiento), diagramas UML y arquitectura software (patrón MVC).

Contribución al perfil profesional: El perfil de egreso en el grado se plantea como el de un profesional capaz de diseñar, desarrollar, implementar y evaluar sistemas inteligentes. La programación orientada a objetos es el paradigma de facto en los principales entornos de desarrollo de aplicaciones de Inteligencia Artificial. Dominar el diseño OO —clases, herencia, encapsulación, patrones de diseño y testing— capacita al egresado o egresada para producir software de IA mantenible, extensible y reutilizable: habilidades directamente demandadas en roles profesionales como ingeniero/a de IA, desarrollador/a de sistemas inteligentes y científico/a de datos.

REQUISITOS Y/O RECOMENDACIONES PARA CURSAR LA ASIGNATURA

Requisito previo

Haber cursado «Fundamentos de Programación» del Grado en Ingeniería en Inteligencia Artificial de la UNED. Esta asignatura proporciona las bases de programación estructurada: estructuras de control (secuencia, selección, iteración), funciones, tipos de datos y colecciones, manejo de ficheros y gestión básica de errores.

No se requiere experiencia previa en programación orientada a objetos. La asignatura está pensada para estudiantes que abordan el paradigma OO por primera vez.

Conocimientos recomendados

•Dominio de la programación estructurada en cualquier lenguaje.

•Familiaridad con algún entorno de desarrollo: Google Colab, Visual Studio Code, IDLE o similar.

•Capacidad de leer documentación técnica básica en inglés: parte de la documentación oficial de las herramientas empleadas (pytest, Sphinx) está disponible solo en inglés.

EQUIPO DOCENTE

Nombre y Apellidos	JUAN MANUEL CIGARRAN RECUERO (Coordinador/a de asignatura)
Correo Electrónico	juanci@lsi.uned.es
Teléfono	91398-9828
Facultad	ESCUELA TÉCN.SUP INGENIERÍA INFORMÁTICA
Departamento	LENGUAJES Y SISTEMAS INFORMÁTICOS

Nombre y Apellidos	JOSE LUIS DELGADO LEAL
Correo Electrónico	jdelgado@lsi.uned.es
Teléfono	91398-8736
Facultad	ESCUELA TÉCN.SUP INGENIERÍA INFORMÁTICA
Departamento	LENGUAJES Y SISTEMAS INFORMÁTICOS

HORARIO DE ATENCIÓN AL ESTUDIANTE

Las/os estudiantes pueden contactar con las/os profesoras/es para resolver dudas sobre la asignatura en primer lugar a través del foro de la asignatura en el campus virtual correspondiente, a través del correo electrónico o por teléfono en el horario que se indica. Si se desea una entrevista personal, debe concertarse previamente. En todo tipo de comunicación con el profesorado se deberá indicar la asignatura a la que se refiere y utilizar el correo de la UNED.

•Juan Manuel Cigarrán Recuero

•Horario de atención: Jueves de 10 a 14 horas.

•Dirección postal: Dpto. de Lenguajes y Sistemas Informáticos. E.T.S.I. Informática. UNED. C/ Juan del Rosal, 16. 2ª planta. Despacho 2.13. 28040 MADRID

•Teléfono: 91.398.9828

•Correo electrónico: juanci@lsi.uned.es

•Jose Luis Delgado Leal

•Horario de atención: Miercoles de 15 a 19 horas.

•Dirección postal: Dpto. de Lenguajes y Sistemas Informáticos. E.T.S.I. Informática. UNED. C/ Juan del Rosal, 16. 2ª planta. Despacho 2.01. 28040 MADRID

•Teléfono: 91.398.8736
•Correo electrónico: jdelgado@lsi.uned.es

TUTORIZACIÓN EN CENTROS ASOCIADOS

En el enlace que aparece a continuación se muestran los centros asociados y extensiones en las que se imparten tutorías de la asignatura. Estas pueden ser:

•**Tutorías de centro o presenciales**: se puede asistir físicamente en un aula o despacho del centro asociado.

•**Tutorías campus/intercampus**: se puede acceder vía internet.

Consultar horarios de tutorización de la asignatura 71031062

COMPETENCIAS QUE ADQUIERE EL ESTUDIANTE

Consultar el apartado de Resultados de Apreindizaje

RESULTADOS DE APRENDIZAJE

Esta asignatura contribuye a los siguientes Resultados de Aprendizaje del Grado en Ingeniería en Inteligencia Artificial:

•**RA06** (Habilidades / Destrezas). Al finalizar el aprendizaje, el o la estudiante será capaz de analizar las necesidades en algorítmica, complejidad computacional, programación, sistemas operativos, bases de datos, estructura, interconexión de sistemas informáticos necesarios para la resolución de problemas de ciencias e ingeniería, de acuerdo con los principios de calidad, fiabilidad y seguridad necesarios.

•**RA08** (Conocimientos / Contenidos). Será capaz de demostrar conocimiento de los procedimientos algorítmicos básicos y los tipos de datos de las tecnologías informáticas necesarios para la resolución de los problemas de inteligencia artificial.

•**RA09** (Habilidades / Destrezas). Será capaz de identificar los algoritmos, estructuras de datos, paradigmas de la programación y lenguajes más adecuados para asegurar la fiabilidad, seguridad y calidad de aplicaciones en problemas que requieran una solución de inteligencia artificial.

Resultados de aprendizaje operativos

Concretados en el ámbito de esta asignatura, al terminar el semestre el estudiante será capaz de:

•**R1.** Explicar los fundamentos del paradigma orientado a objetos —clase, objeto, encapsulación, herencia, polimorfismo y abstracción— y justificar sus ventajas frente a la programación estructurada. *(RA08)*

•**R2.** Implementar en Python 3 jerarquías de clases con encapsulación, herencia simple y múltiple, polimorfismo y abstracción (MRO, super(), abc, Protocol). *(RA08, RA09)*

- **R3.** Diseñar soluciones orientadas a objetos de calidad aplicando los principios SOLID y patrones de diseño básicos (Factory Method, Strategy, Observer, Adapter), con diagramas de clases. *(RA06, RA09)*
- **R4.** Realizar pruebas y depuración de programas orientados a objetos con pytest y el depurador de VS Code/pdb, identificando y corrigiendo errores de forma sistemática. *(RA06, RA09)*
- **R5.** Producir documentación técnica de calidad: docstrings (PEP 257), anotaciones de tipos (PEP 484) y documentación generada con Sphinx o pdoc. *(RA06)*
- **R6.** Abordar proyectos orientados a objetos de complejidad media integrando diseño, implementación, pruebas y documentación en un repositorio Git con entorno virtual. *(RA06, RA09)*
- **R7.** Identificar los paradigmas y lenguajes más adecuados para aplicaciones de Inteligencia Artificial, valorando el papel del paradigma orientado a objetos en ese contexto. *(RA08, RA09)*

CONTENIDOS

Contenidos de la asignatura

Los contenidos de la asignatura se organizan en seis bloques temáticos, desarrollados a lo largo de 17 temas. Cada bloque parte de los conocimientos del anterior, de modo que el estudiante avanza de los conceptos básicos del paradigma orientado a objetos hasta la construcción de un proyecto de software completo. La dedicación estimada por bloque es: 25 + 35 + 40 + 18 + 14 + 16 horas de trabajo del estudiante, más 2 horas de prueba presencial, hasta sumar las 150 horas (6 ECTS) de la asignatura.

1. **Bloque I —Introducción**

   Temas 1 a 3 ·25 horas ·resultados de aprendizaje RA06 y RA08.

   El primer bloque presenta una forma distinta de concebir los programas. Hasta ahora el estudiante ha aprendido a programar describiendo paso a paso lo que el ordenador debe hacer; aquí descubre la **programación orientada a objetos**, una manera de organizar el código en torno a «objetos» que representan elementos del problema real (un cliente, una factura, un vehículo) y que agrupan tanto la información como las acciones que les corresponden.

   Se explica por qué este enfoque se ha impuesto en la industria: cuando los programas crecen, organizarlos en objetos los hace más fáciles de entender, de modificar y de reutilizar. En este bloque también se preparan las herramientas de trabajo que se usarán durante todo el curso. Al terminarlo, el estudiante es capaz de escribir sus primeros programas con este nuevo enfoque y comprende qué aporta frente a lo aprendido anteriormente.

2. **Bloque II —Fundamentos**

   Temas 4 a 7 ·35 horas ·resultados de aprendizaje RA06, RA08 y RA09.

   Es el corazón de la asignatura. Aquí se estudian las cuatro ideas esenciales sobre las que se sostiene todo el paradigma. La primera consiste en **proteger la información** de cada objeto para que solo pueda manipularse de forma controlada, lo que evita muchos errores. La segunda permite **crear objetos a partir de otros ya existentes**, aprovechando lo que tienen en común sin repetir trabajo. La tercera hace posible que distintos objetos **respondan de manera adecuada a una misma orden**, lo que da gran flexibilidad al programa. La cuarta enseña a **describir lo esencial de un objeto** dejando de lado los detalles innecesarios.

   La consecuencia de dominar estos fundamentos es directa: el estudiante pasa de escribir pequeños programas a construir aplicaciones bien estructuradas, en las que un cambio en una parte no obliga a rehacer las demás. También se aprende a gestionar de forma ordenada las situaciones imprevistas o erróneas que pueden surgir mientras un programa se ejecuta.

3. **Bloque III —Diseño de programas orientados a objetos**

   Temas 8 a 11 ·40 horas ·resultado de aprendizaje RA09.

   Saber programar con objetos no garantiza que el resultado sea un buen programa. Este bloque, el de mayor dedicación, enseña a **diseñar software de calidad**: código que otras personas (o uno mismo meses después) puedan entender y modificar sin dificultad. Para ello se presentan una serie de principios ampliamente aceptados por la profesión y un conjunto de **soluciones probadas a problemas frecuentes** —los llamados patrones de diseño—, que se introducen a nivel inicial.

   Se aprende también a reconocer las señales de un diseño mejorable y a corregirlo de forma segura, así como a guardar y recuperar la información de los programas para que no se pierda al cerrarlos. La consecuencia de aplicar estos contenidos es un software más robusto, más fácil de mantener y mejor preparado para crecer. El tratamiento sistemático de estos temas se amplía en asignaturas posteriores del Grado.

4. **Bloque IV —Depuración**

   Temas 12 y 13 ·18 horas ·resultados de aprendizaje RA06 y RA09.

   Todo programa contiene errores; la diferencia está en saber encontrarlos y corregirlos con método. En este bloque el estudiante aprende a **investigar por qué un programa no se comporta como debería**, utilizando las herramientas que permiten detener su ejecución y observar qué ocurre en su interior paso a paso. Además se introduce la idea de **comprobar automáticamente** que el código hace lo que se espera, mediante pruebas que el propio programa puede repetir cuantas veces sea necesario.

   La consecuencia es importante: en lugar de confiar en que «parece que funciona», el estudiante adquiere la disciplina de verificar su trabajo, gana confianza al introducir

cambios y detecta los fallos antes de que lleguen al usuario final.

5. **Bloque V —Documentación**

   Temas 14 y 15 ·14 horas ·resultado de aprendizaje RA06.

   Un programa no solo debe funcionar: también debe poder entenderse. Este bloque trata la **documentación como parte del propio producto de software**, no como un añadido opcional. El estudiante aprende a explicar dentro del código qué hace cada parte y cómo utilizarla, siguiendo las convenciones que la comunidad profesional emplea, y a **generar de forma automática** manuales consultables a partir de esas explicaciones.

   La consecuencia de documentar bien es que el trabajo se vuelve comprensible y reutilizable por otras personas, una destreza imprescindible en cualquier proyecto realizado en equipo o destinado a perdurar en el tiempo.

6. **Bloque VI —Técnicas avanzadas e integración**

   Temas 16 y 17 ·16 horas ·resultado de aprendizaje RA09.

   El bloque final tiene un doble propósito. Por un lado, asoma a algunas **posibilidades más avanzadas** del lenguaje, presentadas como una introducción que despierta interés sin exigir un dominio completo; son recursos que el estudiante encontrará en herramientas profesionales y que comprenderá mejor con la experiencia. Por otro lado, y sobre todo, este bloque **integra todo lo aprendido durante el semestre** en un proyecto completo.

   La consecuencia es que el estudiante comprueba que los contenidos de la asignatura no son piezas aisladas, sino que se combinan para construir una aplicación real, bien organizada y de calidad. Este proyecto sirve además de repaso global de cara a la prueba presencial.

Bloque I —Introducción

El primer bloque presenta una forma distinta de concebir los programas. Hasta ahora el estudiante ha aprendido a programar describiendo paso a paso lo que el ordenador debe hacer; aquí descubre la **programación orientada a objetos**, una manera de organizar el código en torno a «objetos» que representan elementos del problema real (un cliente, una factura, un vehículo) y que agrupan tanto la información como las acciones que les corresponden.

Se explica por qué este enfoque se ha impuesto en la industria: cuando los programas crecen, organizarlos en objetos los hace más fáciles de entender, de modificar y de reutilizar. En este bloque también se preparan las herramientas de trabajo que se usarán durante todo el curso. Al terminarlo, el estudiante es capaz de escribir sus primeros programas con este nuevo enfoque y comprende qué aporta frente a lo aprendido anteriormente.

T1. Del paradigma estructurado al paradigma OO

T2. Entorno de trabajo: Google Colab y Visual Studio Code

T3. Python orientado a objetos: sintaxis básica

Bloque II —Fundamentos

Es el corazón de la asignatura. Aquí se estudian las cuatro ideas esenciales sobre las que se sostiene todo el paradigma. La primera consiste en **proteger la información** de cada objeto para que solo pueda manipularse de forma controlada, lo que evita muchos errores. La segunda permite **crear objetos a partir de otros ya existentes**, aprovechando lo que tienen en común sin repetir trabajo. La tercera hace posible que distintos objetos **respondan de manera adecuada a una misma orden**, lo que da gran flexibilidad al programa. La cuarta enseña a **describir lo esencial de un objeto** dejando de lado los detalles innecesarios.
La consecuencia de dominar estos fundamentos es directa: el estudiante pasa de escribir pequeños programas a construir aplicaciones bien estructuradas, en las que un cambio en una parte no obliga a rehacer las demás. También se aprende a gestionar de forma ordenada las situaciones imprevistas o erróneas que pueden surgir mientras un programa se ejecuta.

T4. Encapsulación y gestión de atributos

T5. Herencia y polimorfismo

T6. Clases abstractas e interfaces

T7. Manejo avanzado de excepciones

Bloque III —Diseño de programas orientados a objetos

Saber programar con objetos no garantiza que el resultado sea un buen programa. Este bloque, el de mayor dedicación, enseña a **diseñar software de calidad**: código que otras personas (o uno mismo meses después) puedan entender y modificar sin dificultad. Para ello se presentan una serie de principios ampliamente aceptados por la profesión y un conjunto de **soluciones probadas a problemas frecuentes** —los llamados patrones de diseño—, que se introducen a nivel inicial.
Se aprende también a reconocer las señales de un diseño mejorable y a corregirlo de forma segura, así como a guardar y recuperar la información de los programas para que no se pierda al cerrarlos. La consecuencia de aplicar estos contenidos es un software más robusto, más fácil de mantener y mejor preparado para crecer. El tratamiento sistemático de estos temas se amplía en asignaturas posteriores del Grado.

T8. Principios SOLID

T9. Patrones de diseño creacionales y estructurales

T10. Patrones de comportamiento y refactorización

T11. Persistencia de datos y serialización

Bloque IV —Depuración de programas

Todo programa contiene errores; la diferencia está en saber encontrarlos y corregirlos con método. En este bloque el estudiante aprende a **investigar por qué un programa no se comporta como debería**, utilizando las herramientas que permiten detener su ejecución y observar qué ocurre en su interior paso a paso. Además se introduce la idea de **comprobar automáticamente** que el código hace lo que se espera, mediante pruebas que el propio programa puede repetir cuantas veces sea necesario.
La consecuencia es importante: en lugar de confiar en que «parece que funciona», el estudiante adquiere la disciplina de verificar su trabajo, gana confianza al introducir cambios y detecta los fallos antes de que lleguen al usuario final.

T12. Técnicas y herramientas de depuración

T13. Testing con pytest

Bloque V —Documentación

Un programa no solo debe funcionar: también debe poder entenderse. Este bloque trata la **documentación como parte del propio producto de software**, no como un añadido opcional. El estudiante aprende a explicar dentro del código qué hace cada parte y cómo utilizarla, siguiendo las convenciones que la comunidad profesional emplea, y a **generar de forma automática** manuales consultables a partir de esas explicaciones.
La consecuencia de documentar bien es que el trabajo se vuelve comprensible y reutilizable por otras personas, una destreza imprescindible en cualquier proyecto realizado en equipo o destinado a perdurar en el tiempo.

T14. Docstrings y convenciones de documentación

T15. Generación automática de documentación

Bloque VI —Introducción a técnicas avanzadas de diseño e integración

El bloque final tiene un doble propósito. Por un lado, asoma a algunas **posibilidades más avanzadas** del lenguaje, presentadas como una introducción que despierta interés sin exigir un dominio completo; son recursos que el estudiante encontrará en herramientas profesionales y que comprenderá mejor con la experiencia. Por otro lado, y sobre todo, este bloque **integra todo lo aprendido durante el semestre** en un proyecto completo.
La consecuencia es que el estudiante comprueba que los contenidos de la asignatura no son piezas aisladas, sino que se combinan para construir una aplicación real, bien organizada y de calidad. Este proyecto sirve además de repaso global de cara a la prueba presencial.

T16. Técnicas avanzadas de Python OO

T17. Integración: el Proyecto Orientado a Objetos

METODOLOGÍA

La asignatura se imparte íntegramente en modalidad no presencial, siguiendo el modelo metodológico de la UNED. Las 150 horas de trabajo total (6 ECTS ×25 h) se distribuyen en las siguientes actividades formativas:

- **AF1 —Aprendizaje autónomo de contenidos teóricos.** 50 h (0 % presencialidad). Estudio del texto-guía, vídeos cortos y materiales en Ágora.
- **AF2 —Aprendizaje autónomo mediante resolución de problemas.** 34 h (0 %). Resolución de ejercicios propuestos y autoevaluación.
- **AF3.2 —Asistencia a tutorías (Centro Asociado).** 12 h (100 %). Sesiones de tutoría presencial o por videoconferencia.
- **AF4 —Aprendizaje con apoyo docente asíncrono.** 8 h (0 %). Foros y materiales de apoyo en Ágora.
- **AF5.3 —Prueba presencial.** 2 h (100 %). Examen de la asignatura en el Centro Asociado.
- **AF6.1 —Prácticas de laboratorio presenciales.** 4 h (100 %). Sesión de laboratorio en Centro Asociado.
- **AF6.3 —Prácticas de laboratorio no presenciales.** 40 h (0 %). PEC1, PEC2 y Proyecto Integrador Final (trabajo autónomo).
- **TOTAL: 150 horas.**

%P = porcentaje de presencialidad.

Prácticas de laboratorio y evaluación continua (SE2)

Las 40 horas de prácticas no presenciales (AF6.3) se articulan en tres entregas evaluables. El tutor o tutora del Centro Asociado es el responsable de su seguimiento y corrección.

- **PEC1 —Bloques I–II.** Entrega en la semana 7. Peso en SE2: 25 % (= 5 % de la nota final). Clases, herencia, encapsulación, polimorfismo y excepciones en Python.
- **PEC2 —Bloques III–IV.** Entrega en la semana 11. Peso en SE2: 25 % (= 5 % de la nota final). Principios de diseño OO, patrones introductorios, tests básicos con pytest y refactorización.
- **Proyecto Integrador Final —Bloques I–VI.** Entrega en la semana 15. Peso en SE2: 50 % (= 10 % de la nota final). Proyecto completo: jerarquía de clases, principios de diseño, tests (70 % cobertura), documentación Sphinx/pdoc, persistencia y repositorio Git.

Papel del tutor o tutora de Centro Asociado

El tutor o tutora orienta al estudiantado durante todo el semestre. Sus funciones principales son:

- Resolver dudas sobre los contenidos y los ejercicios, tanto en sesiones presenciales como a través de Ágora.

- Hacer seguimiento del avance de las PEC y del Proyecto Integrador Final, preferiblemente dentro del horario habitual de tutorías.
- Corregir y calificar las entregas de SE2 aplicando los criterios publicados por el equipo docente en Ágora.
- Coordinar la sesión de prácticas presenciales de laboratorio (AF6.1, 4 horas). La tutoría de las prácticas se realiza, con carácter preferente, dentro del horario ordinario de tutorías del Centro Asociado. Si la carga lo requiere, el tutor o tutora podrá convocar sesiones específicas adicionales con antelación suficiente.

Recomendaciones para el estudio autónomo

POO es una asignatura eminentemente práctica: los conceptos se asimilan programando, no solo leyendo. Las siguientes recomendaciones ayudarán al estudiante a aprovechar al máximo las 150 horas de trabajo:

- **Combina lectura y código.** Para cada tema, lee primero el capítulo correspondiente del texto-guía y después ejecuta los ejemplos en Google Colab modificando parámetros. La comprensión pasiva no es suficiente.
- **Trabaja con constancia semanal.** La asignatura tiene tres entregas evaluables repartidas a lo largo del semestre. Dejar el trabajo para la última semana hace imposible profundizar en los contenidos. Se recomienda dedicar entre 8 y 10 horas semanales de forma regular.
- **Inicia el Proyecto Integrador desde la semana 1.** El enunciado se publica al inicio del semestre. Empezar a tomar notas y a bocetar el diseño desde el principio alivia la carga de las últimas semanas y mejora la calidad del resultado.
- **Usa los errores como herramienta de aprendizaje.** Python genera mensajes de error detallados. Antes de buscar ayuda, intenta leer e interpretar el traceback completo: la mayoría de los errores frecuentes se resuelven así. Después, usa el depurador de VS Code para inspeccionar el estado del programa.
- **Apóyate en los foros de Ágora y en las tutorías.** Si una duda no se resuelve en 30 minutos de trabajo autónomo, es señal de que hay que pedir ayuda. Participar en los foros también ayuda a consolidar el aprendizaje.
- **Lee la documentación oficial de Python.** Muchas respuestas están en docs.python.org/es/3/. Desarrollar el hábito de consultar la documentación oficial es una competencia profesional imprescindible.
- **Sobre el uso de herramientas de IA generativa:** el uso de asistentes de código puede ser útil para explorar alternativas, pero depender de ellos para escribir el código de las entregas impide el aprendizaje real. Si se usan, debe declararse explícitamente en la entrega (véase B15).

SISTEMA DE EVALUACIÓN

TIPO DE PRUEBA PRESENCIAL

Tipo de examen	Examen mixto
Preguntas test	10
Preguntas desarrollo	3
Duración del examen	120 (minutos)
Material permitido en el examen	

PROHIBIDO CUALQUIER TIPO DE MATERIAL

Criterios de evaluación

SE1 —Prueba presencial (80 % de la nota final)

El examen se realiza en el Centro Asociado en la convocatoria ordinaria de junio o en la extraordinaria de septiembre. Dura 120 minutos y no se permite ningún material de apoyo. Evalúa los contenidos de los Bloques I–VI.

El examen es de tipo mixto y consta de dos partes:

Parte de tipo test (35 % de la nota del examen). 10 preguntas de opción múltiple con 4 opciones cada una, de las que solo una es correcta. Las respuestas incorrectas penalizan: cada error resta puntuación.

Parte de desarrollo (65 % de la nota del examen). 3 preguntas de respuesta abierta: análisis de código Python, diseño de jerarquías de clases o aplicación de un patrón a un enunciado concreto.

% del examen sobre la nota final	80
Nota del examen para aprobar sin PEC	
Nota máxima que aporta el examen a la calificación final sin PEC	
Nota mínima en el examen para sumar la PEC	
Comentarios y observaciones	

PRUEBAS DE EVALUACIÓN CONTINUA (PEC)

¿Hay PEC?	Si

Descripción

Pruebas de Evaluación Continua —PEC (10 % de la nota final)

Las PECs son prácticas de programación entregadas a través de Ágora y corregidas por el tutor o tutora del Centro Asociado. Hay dos PECs a lo largo del semestre.

Descripción

PEC1 — Cubre los Bloques I y II (paradigma OO, clases, herencia, encapsulación, polimorfismo y excepciones en Python). El estudiante entrega un notebook Colab ejecutable, un módulo .py y un breve informe de diseño (máx. 4 páginas).

PEC2 — Cubre los Bloques III y IV (principios SOLID, patrones de diseño, testing con pytest y depuración). El estudiante entrega un repositorio Git con historial de commits, un diagrama UML antes/después y un informe técnico (máx. 6 páginas).

Criterios de evaluación

Cada PEC se califica de 0 a 10. Para superar SE2, ambas PECs deben alcanzar una calificación mínima de **5,0/10**. Si alguna no alcanza ese mínimo, SE2 no queda superada con independencia del resto de calificaciones.

Los criterios detallados de corrección de cada PEC serán publicados por el equipo docente en Ágora con suficiente antelación a la fecha de entrega.

Ponderación en la nota final

PEC1: 25 % de SE2 = **5 % de la nota final.**

PEC2: 25 % de SE2 = **5 % de la nota final.**

Ponderación conjunta de ambas PECs: **10 % de la nota final.**

Fechas de entrega

PEC1: cierre al final de la **semana 7** del semestre (aproximadamente finales de marzo de 2027).

PEC2: cierre al final de la **semana 11** del semestre (aproximadamente finales de abril de 2027).

Ponderación de la PEC en la nota final	Ponderación en la nota final PEC1: 25 % de SE2 = 5 % de la nota final. PEC2: 25 % de SE2 = 5 % de la nota final. Ponderación conjunta de ambas PECs: 10 % de la nota final.

Fecha aproximada de entrega

Comentarios y observaciones

Las entregas se realizan exclusivamente a través de la plataforma Ágora. Las calificaciones de las PECs aprobadas se conservan para la convocatoria extraordinaria de septiembre del mismo curso.

OTRAS ACTIVIDADES EVALUABLES

¿Hay otra/s actividad/es evaluable/s? Si

Descripción

Otras actividades evaluables —Proyecto Integrador Final (10 % de la nota final)

El Proyecto Integrador Final es una práctica de laboratorio (AF6.3) que integra los contenidos de los seis bloques de la asignatura. Se gestiona en Ágora y es corregido y calificado por el tutor o tutora del Centro Asociado.

Descripción

El estudiante desarrolla un proyecto de software orientado a objetos completo basado en las dos PECs realizadas que debe incluir:

Jerarquía de clases con aplicación de los principios SOLID.

Al menos un patrón de diseño justificado.

Batería de tests con pytest (cobertura mínima del 70 %).

Documentación generada automáticamente con Sphinx o pdoc.

Persistencia de datos (JSON, CSV o sqlite3).

Repositorio Git con historial de commits progresivo.

Memoria técnica (máximo 10 páginas).

Criterios de evaluación

Criterios de evaluación

El proyecto se califica de 0 a 10. Para superar SE2, el Proyecto Integrador Final debe alcanzar una calificación mínima de 5,0/10. Si no se alcanza ese mínimo, SE2 no queda superada con independencia de las notas de PEC1 y PEC2.

Los criterios de corrección serán publicados en Ágora al inicio del semestre.

Ponderación en la nota final

El Proyecto Integrador Final representa el 50 % de SE2, es decir, el 10 % de la nota final.

Fecha de entrega

Cierre al final de la semana 15 del semestre (aproximadamente mediados de mayo de 2027).

Ponderación en la nota final	El Proyecto Integrador Final representa el 50 % de SE2, es decir, el 10 % de la nota final.
Fecha aproximada de entrega	
Comentarios y observaciones	

El proyecto se realizará de forma individual, según las instrucciones publicadas por el tutor o tutora en Ágora al inicio del semestre. La calificación obtenida se conserva para la convocatoria extraordinaria de septiembre del mismo curso, salvo renuncia expresa por escrito.

¿CÓMO SE OBTIENE LA NOTA FINAL?

La nota final de la asignatura se obtiene combinando la **prueba presencial (SE1)** vale el **80 %** y las **prácticas de laboratorio y PECs (SE2)** el **20 %**. La nota final solo se calcula si SE2 está superada.

Fórmula de la nota final:

Nota final = 0,80 ×SE1 + 0,05 ×PEC1 + 0,05 ×PEC2 + 0,10 ×Proyecto Integrador Final

Condición: SE2 debe estar superada (las tres entregas con 5,0/10). Si SE2 no está superada, la nota final sera de suspenso independientemente de la calificacion obtenida en el examen.

BIBLIOGRAFÍA BÁSICA

Las referencias que figuran a continuación se registrarán en la base de datos de bibliografía de la UNED. Se incluye una nota de uso para orientar al estudiante. Formato APA 7.ª ed.

1. Boucheny, V. (2025). *Aprender Programación Orientada a Objetos con Python: con ejercicios prácticos y corregidos* (3.ª ed.). Ediciones ENI. ISBN 978-2-409-05152-4. *Manual principal de la asignatura. Cubre el ciclo completo de la POO con Python 3, patrones de diseño elementales y buenas prácticas, con ejercicios resueltos.*
2. Marzal Varó, A., Gracia Luengo, I. y García Sevilla, P. (2014). *Introducción a la programación con Python 3*. Universitat Jaume I. ISBN 978-84-697-1178-1. Acceso abierto: repositori.uji.es

Texto complementario gratuito en español. Recomendado para reforzar los fundamentos del Bloque I.

BIBLIOGRAFÍA COMPLEMENTARIA

1. Martin, R. C. (2012). *Código limpio: Manual de estilo para el desarrollo ágil de software*. Anaya Multimedia. ISBN 978-84-415-3210-6.
   Referencia sobre código limpio y principios SOLID. Bloques III y IV.
2. Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (2002). *Patrones de Diseño: Elementos de software orientado a objetos reutilizable*. Addison Wesley / Pearson Educación. ISBN 978-84-7829-059-8.
   Obra fundacional de los patrones GoF. Bloque III.
3. Hunt, A. y Thomas, D. (2022). *El programador pragmático: Edición especial. Viaje a la maestría*. Anaya Multimedia. ISBN 978-84-415-4587-8.
   Edición española actualizada. Bloques III y VI.
4. Okken, B. (2022). *Python Testing with pytest* (2.ª ed.). Pragmatic Bookshelf. ISBN 978-1-68050-860-4. *[Solo en inglés]*
   Guía completa de pytest y pytest-cov. Bloque IV.

RECURSOS DE APOYO Y WEBGRAFÍA

Recursos en línea complementarios, de acceso libre y gratuito. URLs verificadas en mayo de 2026.

Python y buenas prácticas

•Documentación oficial de Python 3 en español: https://docs.python.org/es/3/

•PEP 8 —Guía de estilo para código Python: https://peps.python.org/pep-0008/

•PEP 257 —Convenciones para docstrings: https://peps.python.org/pep-0257/

•PEP 484 —Anotaciones de tipo: https://peps.python.org/pep-0484/

Testing y documentación

•Documentación oficial de pytest: https://docs.pytest.org/

•Documentación oficial de Sphinx: https://www.sphinx-doc.org/

Entornos de desarrollo

•Google Colaboratory: https://colab.research.google.com/

•Visual Studio Code: https://code.visualstudio.com/

Recurso pedagógico complementario

•Python for Everybody —versión en español: https://es.py4e.com/

IGUALDAD DE GÉNERO

En coherencia con el valor asumido de la igualdad de género, todas las denominaciones que en esta Guía hacen referencia a órganos de gobierno unipersonales, de representación, o miembros de la comunidad universitaria y se efectúan en género masculino, cuando no se hayan sustituido por términos genéricos, se entenderán hechas indistintamente en género femenino o masculino, según el sexo del titular que los desempeñe.