

# Tecnatura Universitaria en Programación Informática

## Universidad Nacional de Quilmes

### Índice

<b>1. Identificación de la carrera</b>	<b>3</b>
1.1. Fundamentación . . . . .	3
1.2. Denominación . . . . .	3
1.3. Nivel . . . . .	4
1.4. Ubicación en la estructura institucional . . . . .	4
<b>2. Horizontes de la carrera</b>	<b>4</b>
2.1. Objetivos . . . . .	4
2.2. Perfil del egresado . . . . .	5
2.3. Alcances . . . . .	5
<b>3. Diseño curricular de la carrera</b>	<b>6</b>
3.1. Duración estimada . . . . .	6
3.2. Requisitos de ingreso . . . . .	6
3.3. Estructura curricular . . . . .	6
3.3.1. Materias del núcleo básico obligatorio . . . . .	6
3.3.2. Materias del núcleo avanzado obligatorio . . . . .	6
3.3.3. Materias del núcleo complementario . . . . .	7
3.3.4. Trabajo de inserción profesional . . . . .	8
3.3.5. Otros requisitos . . . . .	8
3.3.6. Asignación horaria total . . . . .	8
3.3.7. Secuencialidad del recorrido curricular . . . . .	8
3.4. Objetivos y contenidos mínimos de las materias del núcleo básico obligatorio	10
3.4.1. Introducción a la Programación . . . . .	11
3.4.2. Organización de Computadoras . . . . .	12
3.4.3. Matemática I . . . . .	13
3.4.4. Programación con Objetos I . . . . .	14
3.4.5. Bases de Datos . . . . .	16
3.4.6. Estructuras de Datos . . . . .	17
3.4.7. Programación con Objetos II . . . . .	19
3.5. Objetivos y contenidos mínimos de las materias del núcleo avanzado obligatorio . . . . .	20
3.5.1. Redes de Computadoras . . . . .	21
3.5.2. Sistemas Operativos . . . . .	22
3.5.3. Programación Concurrente . . . . .	23
3.5.4. Matemática II . . . . .	24
3.5.5. Elementos de Ingeniería de Software . . . . .	25
3.5.6. Construcción de Interfaces de Usuario . . . . .	27
3.5.7. Estrategias de Persistencia . . . . .	29
3.5.8. Programación Funcional . . . . .	31

---

3.5.9.	Desarrollo de Aplicaciones . . . . .	33
3.5.10.	Laboratorio de Sistemas Operativos y Redes . . . . .	35
3.6.	Contenidos mínimos de los niveles de idioma inglés . . . . .	36
3.6.1.	Inglés I . . . . .	36
3.6.2.	Inglés II . . . . .	36
3.7.	Contenidos mínimos de las materias del núcleo complementario . . . . .	37
3.7.1.	Seguridad Informática . . . . .	37
3.7.2.	Bases de Datos II . . . . .	37
3.7.3.	Participación y Gestión en Proyectos de Software Libre . . . . .	37
3.7.4.	Introducción a las Arquitecturas de Software . . . . .	38
3.7.5.	Programación con Objetos III . . . . .	38
3.7.6.	Introducción a la Bioinformática . . . . .	38
3.7.7.	Políticas Públicas en la Sociedad de la Información y la Era Digital . . . . .	38
3.7.8.	Sistemas de Información Geográfica . . . . .	39
3.7.9.	Herramientas Declarativas en Programación . . . . .	39
3.7.10.	Introducción al Desarrollo de Videojuegos . . . . .	39
3.7.11.	Derechos de Autor y Derecho de Copia en la Era Digital . . . . .	40
3.7.12.	Seminarios . . . . .	40
3.7.13.	Seminarios sobre Herramientas o Técnicas Puntuales . . . . .	40
<b>4.</b>	<b>Cuadro resumen</b>	<b>41</b>

---

# 1. Identificación de la carrera

## 1.1. Fundamentación

La Argentina muestra una actividad económica robusta en el área de desarrollo de software, originada en y retroalimentándose con una cultura informática temprana y ampliamente extendida, al menos en los principales centros urbanos.

El aumento sostenido que se espera en la demanda global de servicios asociados a las tecnologías de la información y las comunicaciones (TICs) augura para el área un amplio potencial de crecimiento.

El país cuenta con varios de los factores necesarios para aprovechar este potencial en particular respecto del desarrollo de software, entre ellos una amplia base de empresas del sector de distintas características y tamaños que trabajan tanto en el mercado local como en el internacional, una cantidad interesante de profesionales con capacidades competitivas a nivel global, y un fuerte y consistente apoyo estatal al sector.

Por otro lado, Argentina no es el único país que ha detectado la posibilidad de generación de empleos de calidad y de desarrollo económico que brindan las TICs en general y el desarrollo de software en particular. Hay varios países que vienen desarrollando estrategias que les han permitido una inserción importante en los mercados mundiales dentro de estas actividades.

Creemos que el desarrollo del sector en la Argentina puede beneficiarse de un posicionamiento global que destaque la capacidad de proveer servicios de alta calidad. Esta visión motiva la orientación del plan que proponemos, que aspira a conjugar práctica extensiva en habilidades directamente relacionadas con las necesidades que percibimos en el mercado laboral con una sólida formación en los conceptos de base de la programación y con el énfasis en el cuidado de distintos criterios de calidad de los productos de software construidos.

Otro factor que estimamos importante destacar es el entorno sociohumano de la UNQ. La Universidad está inserta en un área urbana densamente poblada con una gran dispersión en la calidad de la educación recibida por los jóvenes que ingresan en la Universidad.

La propuesta que presentamos tiene en cuenta la realidad de la población de estudiantes con que contamos. El plan está pensado para una transmisión gradual y progresiva de los conceptos principales que deben ser incorporados, incluyendo además una fuerte carga horaria de las materias de los dos primeros cuatrimestres que permite realizar un seguimiento en la adquisición de hábitos de estudio y en la comprensión de los principios básicos de la programación. De esta forma se hace énfasis en el acompañamiento a cada estudiante en su incorporación a una currícula de nivel universitario.

La propuesta que presentamos aspira a fortalecer la capacidad de los egresados/as de ocupar empleos de calificación media o alta en un sector que cuenta con una amplia oferta laboral y buenas perspectivas de crecimiento, tanto a nivel global como en particular para las áreas metropolitanas de Buenos Aires y La Plata en zonas fácilmente accesibles desde el área de influencia de la Universidad, y que genera empleos que por lo general tienen buenas remuneraciones y condiciones razonables de trabajo. De esta forma, buscamos brindar a la comunidad de la zona de inserción de la UNQ una alternativa laboral de calidad.

De acuerdo a lo antedicho, presentamos esta propuesta de nuevo plan de estudios en el convencimiento que fortalece el cumplimiento de los objetivos trazados en el programa que dio origen a la carrera.

## 1.2. Denominación

Carrera: Tecnicatura Universitaria en Programación Informática.

Título: Técnico Universitario en Programación Informática.

---

### 1.3. Nivel

Pregrado.

### 1.4. Ubicación en la estructura institucional

La carrera tiene una estructura autónoma inserta en el Departamento de Ciencia y Tecnología de la UNQ.

## 2. Horizontes de la carrera

### 2.1. Objetivos

Formar técnicos/as capaces de elucidar e implementar soluciones en un amplio espectro de problemas asociados a las tareas de diseño/programación dentro del desarrollo de software, en un alcance razonable para un egresado/a pre-universitario, siendo capaces de aprovechar los conceptos aprehendidos en la carrera para pensar y resolver situaciones concretas, y basados en una amplia experiencia práctica obtenida durante el recorrido de la carrera.

En particular se espera que un egresado/a posea las siguientes capacidades:

- concebir una solución, implementarla y describir los conceptos que fundamentan las decisiones que tomó, ante un problema concreto de diseño y/o programación de complejidad mediana.
- construir programas informáticos teniendo en cuenta parámetros básicos de calidad (grado de test, claridad, mantenibilidad, robustez, extensibilidad) en varios lenguajes de programación, e incorporar nuevos lenguajes y estilos de programación al marco de los conceptos que conoce.
- manejar con fluidez el entorno que necesita un programador para trabajar: sistema operativo, entornos de desarrollo, entornos de ejecución.
- tener elementos que le faciliten el trabajo en grupo, tanto en lo actitudinal (compartir conocimientos, privilegiar colaboración a competencia, organizar tareas) como en lo técnico (conocer herramientas y entornos).
- incorporar a su práctica nuevas herramientas que vayan apareciendo en el ámbito profesional.
- comprender que las actividades de programación se inscriben muchas veces en un marco más amplio de proyectos de desarrollo de software, y cuáles son sus roles específicos dentro de un equipo de proyecto.

---

## 2.2. Perfil del egresado

El egresado (o la egresada) es un técnico universitario cuya área de acción principal es la problemática de la construcción de software, que se corresponde con las tareas tradicionalmente conocidas como diseño y programación/codificación.

El recorrido de la carrera le brinda una especialización en proyectos de desarrollo de aplicaciones organizacionales utilizando el paradigma de objetos, en este ámbito específico tiene los conocimientos para insertarse rápida y satisfactoriamente en el mercado laboral.

De acuerdo al perfil propuesto, el egresado/a deberá:

Tener una base conceptual sólida que le permita participar en proyectos de desarrollo de software de distinta índole, tanto respecto del tipo de software como de las herramientas de desarrollo utilizadas; y también adaptarse a las nuevas herramientas que van apareciendo en el ámbito laboral.

Comprender adecuadamente la pertinencia de construir software de acuerdo a varios parámetros de calidad, entre los que destacamos: claridad, inclusión de tests automáticos extensivos, extensibilidad, robustez frente a fallos, uso eficiente de recursos; también manejar los principales conceptos y herramientas requeridos para que sus productos cuenten con grados adecuados de calidad. Asimismo comprender la conveniencia de valorar y tener en cuenta los conceptos de estándares abiertos y software libre en los entornos operativos y herramientas de desarrollo que se utilizan.

Contar con conocimientos que le permitan asumir otras tareas además de la construcción (elucidación de requerimientos, despliegue, administración del entorno de ejecución) para proyectos de porte pequeño.

## 2.3. Alcances

Si bien los alcances del Técnico Universitario en Programación Informática son los que se enuncian a continuación, la responsabilidad primaria y la toma de decisiones la ejerce en forma individual y exclusiva el poseedor del título con competencias reservadas según el régimen del artículo 43 de la Ley de Educación Superior.

La Tecnicatura debe formar egresados capaces de participar en el desarrollo de proyectos de software de cualquier porte y casi cualesquiera características, adaptándose a distintos tipos de proyecto, formas de trabajo y herramientas. El grado de esta participación dependerá de las características de cada proyecto.

En particular, se espera que un egresado pueda

- participar en todas las actividades de desarrollo e implantación, incluyendo la elección de las herramientas a utilizar, para proyectos de desarrollo de software de pequeño porte (i.e. cuya magnitud no supera los meses/hombre y que no reviste características críticas o inusualmente complejas), en particular la construcción de aplicaciones organizacionales.
- participar en tareas específicas de diseño/codificación de software, para proyectos de mediano porte (en el orden de 1/2 años-hombre y que no cuenten con características particularmente críticas o complejas)
- formar parte del equipo de desarrollo en roles que no requieran capacidades de decisión en cuestiones relevantes, para proyectos de mayor porte y/o criticidad

El egresado también podrá colaborar con la administración de redes de computadoras en entornos que no cuenten con una dimensión o grado de complejidad importantes, adaptándolos a las necesidades del lugar en donde se encuentren.

---

### 3. Diseño curricular de la carrera

#### 3.1. Duración estimada

Seis cuatrimestres, en los que se incluyen tanto las materias como el trabajo de inserción profesional.

#### 3.2. Requisitos de ingreso

Los establecidos la Ley 24521 de Educación Superior, o las leyes que eventualmente la reemplacen.

#### 3.3. Estructura curricular

Para acceder al Título de Técnico Universitario en Programación Informática, el estudiante deberá: cursar la totalidad de las materias de los núcleos básico obligatorio y avanzado obligatorio, obtener 24 créditos en materias del núcleo complementario y realizar un trabajo de inserción profesional.

##### 3.3.1. Materias del núcleo básico obligatorio

Las materias del núcleo básico obligatorio brindan al estudiante la formación requerida en los conceptos básicos de la programación y brindan conocimientos fundamentales de áreas anexas y de matemática; conocimientos y formación requeridos para abordar el resto de la carrera.

Para todas las materias incluidas en la siguiente tabla, el régimen de cursado es cuatrimestral, y la modalidad es presencial.

Materia	Horas semanales	Carga horaria total	Créditos
Introducción a la Programación	8	144	16
Organización de Computadoras	6	108	12
Matemática I	8	144	16
Programación con Objetos I	8	144	16
Bases de Datos	6	108	12
Estructuras de Datos	8	144	16
Programación con Objetos II	6	108	12
<b>Totales</b>		900	102

##### 3.3.2. Materias del núcleo avanzado obligatorio

Las materias del núcleo avanzado obligatorio completan la formación obligatoria del estudiante.

Para todas las materias incluidas en la siguiente tabla, el régimen de cursado es cuatrimestral, y la modalidad es presencial.

---

<b>Materia</b>	<b>Horas semanales</b>	<b>Carga horaria total</b>	<b>Créditos</b>
Redes de Computadoras	6	108	12
Sistemas Operativos	6	108	12
Programación Concurrente	4	72	8
Matemática II	4	72	8
Elementos de Ingeniería de Software	6	108	12
Construcción de Interfaces de Usuario	6	108	12
Estrategias de Persistencia	6	108	12
Programación Funcional	4	72	8
Desarrollo de Aplicaciones	6	108	12
Laboratorio de Sistemas Operativos y Redes	4	72	8
<b>Totales</b>		936	104

### 3.3.3. Materias del núcleo complementario

Las materias del núcleo complementario permiten orientar al estudiante hacia un perfil determinado dentro del universo amplio y cambiante de los proyectos de desarrollo de software.

<b>Materia</b>	<b>Horas semanales</b>	<b>Carga horaria total</b>	<b>Créditos</b>
Seguridad Informática	4	72	8
Bases de Datos II	4	72	8
Participación y Gestión en Proyectos de Software Libre	4	72	8
Introducción a las Arquitecturas de Software	4	72	8
Programación con Objetos III	4	72	8
Introducción a la Bioinformática	4	72	8
Políticas Públicas en la Sociedad de la Información y la Era Digital	4	72	8
Sistemas de Información Geográfica	4	72	8
Herramientas Declarativas en Programación	4	72	8
Introducción al Desarrollo de Videojuegos	4	72	8
Derechos de Autor y Derecho de Copia en la Era Digital	4	72	8
Seminarios	4	72	8
Seminarios sobre Herramientas o Técnicas Puntuales		32	4
<b>Totales</b>		216	24

Para todas las materias incluidas en la siguiente tabla la modalidad es presencial.

Las materias indicadas con una carga total de 32 horas podrán desarrollarse durante todo un cuatrimestre o ser más intensivos y ocupar sólo parte del mismo, de ahí que no se indique una cantidad de horas semanales para los mismos. Para las materias indicadas con una carga horaria total de 72 horas, el régimen de cursado es cuatrimestral.

---

### 3.3.4. Trabajo de inserción profesional

En esta actividad el estudiante, a través de la realización de un trabajo acorde al perfil profesional del egresado, demostrará la integración de los conocimientos adquiridos en las diferentes materias en el desarrollo concreto de un pequeño proyecto de software guiado por un profesor de la carrera.

Este trabajo es de realización individual o en grupos pequeños, y su tiempo de realización no debe extenderse más de un semestre salvo excepciones puntuales. Se deben plantear trabajos de una extensión de entre 100 y 180 horas por estudiante. El trabajo otorga 10 créditos.

La modalidad del trabajo de inserción profesional se instrumentará por normativa específica.

### 3.3.5. Otros requisitos

Para acceder al Título de Técnico Universitario en Programación Informática es necesario, además de la aprobación de las materias y trabajo de desarrollo de software ya mencionados, acreditar conocimientos de Inglés análogos a dos niveles cuatrimestrales de 54 horas cada uno.

### 3.3.6. Asignación horaria total

La carga horaria total para acceder al título de Técnico Universitario en Programación Informática es de 2260 horas, de acuerdo al detalle siguiente.

Núcleo	Carga horaria total mínima	Créditos
Básico obligatorio	900	102
Avanzado obligatorio	936	104
Complementario	216	24
Trabajo de de inserción profesional	100	10
Otros requisitos (niveles de inglés)	108	0
<b>Totales</b>	<b>2260</b>	<b>240</b>

### 3.3.7. Secuencialidad del recorrido curricular

A fin de garantizar coherencia en el recorrido curricular de cada estudiante, se establece que

- Para la inscripción en materias del núcleo avanzado obligatorio es necesario acreditar la aprobación del 70 % de los créditos del núcleo básico obligatorio.
- Para la inscripción en materias del núcleo complementario es necesario acreditar la aprobación del 100 % de los créditos del núcleo básico obligatorio y del 50 % de los créditos del núcleo avanzado obligatorio.

Asimismo, se recomendará fuertemente a los estudiantes que respeten las secuencias indicadas en el gráfico de la Figura 1, que responden a las dependencias entre los conceptos que se introducen en cada una; el gráfico incluye una posible organización en cuatrimestres.



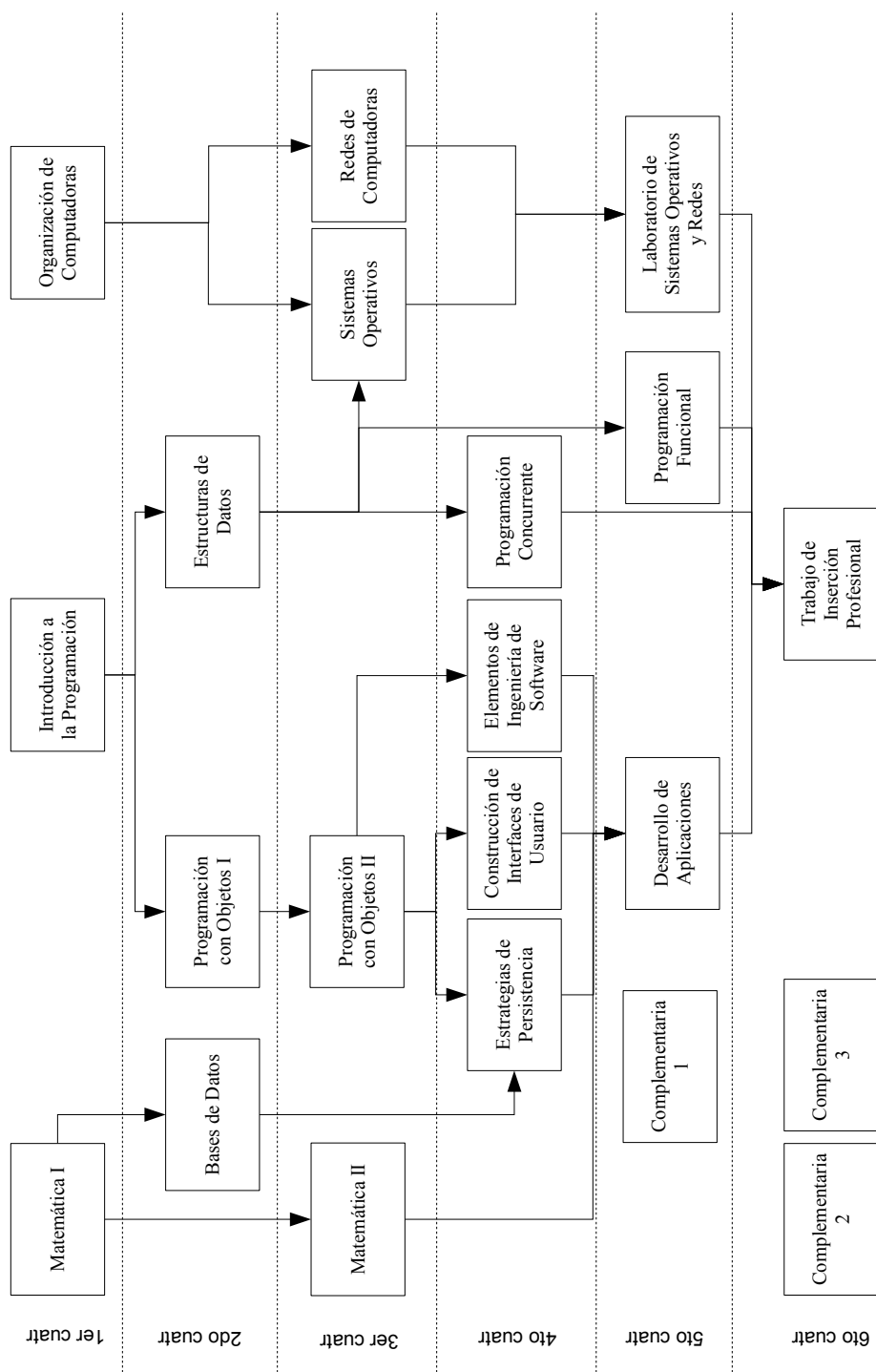


Figura 1: Gráfico de dependencias de la Tecnicatura Universitaria en Programación Informática

---

### **3.4. Objetivos y contenidos mínimos de las materias del núcleo básico obligatorio**

Se consignan, para cada materia del núcleo básico obligatorio, sus objetivos y contenidos mínimos. Asimismo, para algunas asignaturas se indican contenidos sugeridos que complementan adecuadamente los contenidos consignados como mínimos.

---

### 3.4.1. Introducción a la Programación

#### Objetivos

Que el estudiante:

- Pueda pensar, implementar y depurar un programa que resuelva problemas sencillos.
- Entienda las ideas de secuencia y estado, y las pueda usar al pensar sus programas.
- Pueda estructurar su programa en bloques que se invocan entre sí, entienda los conceptos de parámetro y valor de retorno.
- Entienda el concepto de estructura de datos, y pueda aprovecharlo para organizar el estado de los programas que construye.
- Maneje ciertos principios, prácticas y metodologías básicos que resulten en programas más robustos y mantenibles, p.ej. comentarios, elección de nombres, precondiciones, terminación.

#### Contenidos mínimos

- Qué es un programa.
- Las herramientas del programador: entornos de ejecución y de desarrollo.
- Principios de la programación imperativa: acciones y comandos, valores y expresiones, tipos, estado.
- Terminación y parcialidad. Precondiciones como metodología para desarrollo de software robusto.
- Principios de la programación estructurada: funciones y procedimientos. Necesidad de darle una estructura a un programa no trivial.
- Resolución de pequeños problemas mediante programas.
- Estructuras de datos básicas: listas y registros.

---

### 3.4.2. Organización de Computadoras

#### Objetivos

Que el estudiante:

- Entienda los diferentes sistemas de numeración usados típicamente en los computadores.
- Entienda los principios básicos de funcionamiento de las computadoras. Reconocer los componentes funcionales y entender su funcionamiento.
- Entienda el mecanismo de ejecución de los programas.
- Adquiera habilidades para resolver problemas usando lógica digital (circuitos combinatorios y secuenciales) y aplicando las nociones de reuso y modularización.
- Conozca las características básicas de la comunicación de la computadora con el usuario y con otras computadoras.

#### Contenidos mínimos

- Representación de la información: alfanumérico, numérico, punto fijo y flotante, ASCII. Sistema de numeración binario.
- Aritmética de las computadoras: Unidades. Funcionamiento y organización (modelo de Von Neumann).
- Unidades funcionales: Unidad Central de Proceso, Unidad de Control, memorias, ciclo de instrucciones, direccionamiento, subsistema de Memoria. Periféricos: conceptos y principio de funcionamiento. Procesadores de Entrada/Salida.
- Lógica digital: tablas de verdad, equivalencia de fórmulas proposicionales, circuitos combinatorios, circuitos secuenciales
- Arquitectura del computador: Componentes de la CPU, memoria principal y secundaria, jerarquía de memorias.
- Subsistema de Entrada/Salida.
- Lenguaje Máquina. Código fuente y código objeto.

---

### 3.4.3. Matemática I

#### Objetivos

Son objetivos generales:

- Proporcionar algunas bases matemáticas imprescindibles, en concordancia con el área de Programación, de modo de contribuir a la formación integral del Técnico Universitario en Programación Informática.
- Introducir a los estudiantes en los procesos típicos del pensamiento matemático: conjeturar, inducir, deducir, probar, generalizar, particularizar, modelar y representar algunos conceptos.

Son objetivos específicos que el estudiante logre:

- Armar en forma correcta frases en lógica proposicional, formalizar situaciones sencillas en lógica de primer orden e iniciarse en la elaboración de demostraciones en ambas lógicas, cuestiones necesarias para el desarrollo del paradigma lógico de razonamiento.
- Adquirir habilidades en el manejo de demostraciones por inducción matemática que relativa al conjunto de los números naturales, resulta ser la base de la recursividad.
- Operar con conjuntos e interpretar enunciados relativos a los mismos.
- Conocer distintas formas de establecer jerarquías entre los elementos de un conjunto a través de las relaciones de orden.
- Conocer distintas formas de clasificar los elementos de un conjunto, a través de las relaciones de equivalencia.
- En el conjunto de los enteros: operar con el máximo común divisor y mínimo común múltiplo, reconocer la importancia de los números primos en la construcción de los números enteros y resolver ecuaciones con congruencias.
- Reconocer el modo en que reglas útiles para contar, que tienen que ver con el sentido común, se puedan relacionar con distintos problemas de conteo que podrán ser modelizados.

#### Contenidos mínimos

- Lógica proposicional y de primer orden. Técnicas de prueba.
- Teoría básica de conjuntos.
- Relaciones binarias: relaciones de orden, relaciones de equivalencia, relaciones funcionales.
- Aritmética entera y modular.
- Inducción matemática sobre números naturales.
- Elementos básicos de análisis combinatorio.

---

### 3.4.4. Programación con Objetos I

#### Objetivos

Que el estudiante:

- Pueda modelar una situación sencilla, pero no trivial, usando correctamente las nociones básicas del paradigma de objetos: objeto, mensaje, clase, comportamiento, responsabilidad, protocolo.
- Entienda cómo se maneja el estado en el paradigma de objetos, esto es, haciendo que los objetos se conozcan entre sí.
- Conozca la forma básica de manejar el análogo a estructuras de datos, que son las colecciones.
- Entienda qué es el polimorfismo, y pueda usarlo adecuadamente en situaciones concretas. Adquiera cierta pericia en reconocer los tipos que puede tener un objeto, y en pensar en términos de tipo/interface.
- Entienda qué es la herencia, pueda distinguir formas adecuadas e inadecuadas de aprovecharla, y pueda usarla adecuadamente en situaciones concretas.
- Se enfrente a situaciones en las que resultan pertinentes formas de relacionar los objetos menos intuitivas/obvias, conozca el concepto de patrón de diseño, y pueda utilizar algunos patrones en situaciones concretas.
- Adquiera la costumbre de trabajar con testeo automático y unitario, vea que los tests también pueden verse como manuales de uso de los elementos que conforman un programa.
- Adquiera nociones básicas sobre manejo de errores: qué es un error y qué no, qué puede hacerse cuando en un programa se detecta una situación de error.

#### Contenidos mínimos

- Conceptos fundantes del paradigma: objeto y mensaje. Visión externa del objeto: dispositivo computacional capaz de recibir mensajes y otorgar respuestas adecuadas. Relevancia de estos conceptos (con qué objetos cuento, qué mensajes le puedo enviar a cada uno) en el desarrollo de software.
- Concepto de polimorfismo en objetos, comprensión de las ventajas de aprovecharlo.
- Protocolo/interfaz, concepto de tipo en objetos, comprensión de que un objeto puede asumir distintos tipos. La interfaz como contrato al que se comprometen ciertos objetos, posibilidad de reforzar ese contrato.
- Estado en el paradigma de objetos: referencias, conocimiento, estado interno.
- Métodos, clases, herencia, method lookup.
- Conceptos de responsabilidad y delegación, su rol al definir una trama de objetos que responde a requerimientos determinados.
- Colecciones: conceptualización como objetos, caracterización a partir de los conceptos de protocolo y responsabilidad, protocolo, acceso a sus elementos.
- Testeo automático y repetible, test como comprobación tanto del correcto funcionamiento como de que los objetos definidos son efectivamente usables.

---

### Contenidos sugeridos

- Ambientes basados en prototipos.
- Chequeo de tipo: concepto, chequeo dinámico vs. estático.
- Protocolo de Stream / Iterator: open / next / close. Nociones de acceso aleatorio y secuencial a partir de esto.

---

### 3.4.5. Bases de Datos

#### Objetivos

Que el estudiante:

- Entienda qué es un modelo de datos.
- Entienda la diferencia entre pensar a nivel esquema y pensar a nivel instancia.
- Pueda armar un modelo de entidad-relación para un dominio no trivial.
- Pueda armar un modelo relacional para un dominio no trivial.
- Entienda el concepto de definir relaciones sobre una instancia relacional vista como un objeto matemático.
- Pueda resolver consultas lógicamente complejas en el lenguaje SQL, incluyendo joins parciales, consultas anidadas y agrupamientos.
- Entienda el concepto de transacción.

#### Contenidos mínimos

- Qué es un modelo de datos, modelos conceptuales, lógicos y físicos.
- Modelo de entidad-relación: conceptos básicos.
- Modelo relacional: tabla, atributo, dominio, valor, fila; restricciones de integridad; operaciones que se pueden hacer.
- SQL: concepto de lenguaje de consulta, sintaxis, concepto de join, agrupamientos, subqueries, joins parciales.
- Transacción: concepto, demarcación de transacciones.

#### Contenidos sugeridos

- Normalización de esquemas de BD: anomalías de inserción, modificación y borrado; concepto de dependencia funcional, formas normales y relación con las anomalías.
- Algebra relacional: concepto, operaciones, ver a las instancias de tabla como conjuntos de filas y a los esquemas de tabla como conjuntos de atributos.



---

### 3.4.6. Estructuras de Datos

#### Objetivos

Que el estudiante:

- Comprenda la noción de dato y de estructuras de datos, y su importancia e interrelación estrecha con la estructura algorítmica de un programa.
- Entienda la diferencia entre acceso aleatorio y acceso secuencial.
- Conozca la idea de interface de una estructura de datos, y sea capaz de utilizarla productivamente para la solución de problemas.
- Conozca la interface de distintas estructuras de datos básicas (pilas, colas, listas, árboles, hashing, etc.) y las utilice adecuadamente
- Comprenda y utilice la noción de estructura contenedora, y la capacidad de realizar combinaciones complejas utilizándolas (i.e. pila de lista de registros, etc.)
- Se familiarice con las nociones de ámbito y de pasaje de parámetros por valor o referencia.
- Maneje alguno de los principios básicos de diseño de interfases de una estructura de datos (separación en constructores e inspectores de una interfase, ecuaciones entre combinaciones de constructores, etc.), y pueda reconocerlos en situaciones prácticas junto con su utilidad.
- Comprenda el concepto de asignación dinámica de memoria, y pueda hacer programas que hagan un manejo dinámico explícito de memoria en forma adecuada.
- Entienda la noción de implementación de una estructura de datos, y de su eficiencia, y sea capaz de implementar las interfases vistas anteriormente con distintas alternativas variadas en eficiencia.
- Se familiarice con las tareas de compilar y vincular programas para lograr un ejecutable.
- Pueda resolver problemas mediante programas recursivos, y entienda la diferencia entre una resolución recursiva y otra iterativa.

#### Contenidos mínimos

- Recursión sobre listas y árboles. Programas recursivos.
- Tipos algebraicos: maybe, either, enumerativos, listas, árboles binarios, árboles generales.
- Estructuras contenedoras: pilas, colas, diccionarios, heaps, árboles balanceados, contenedores basados en representaciones numéricas.
- Nociones de representación e invariante de representación y su utilidad en el diseño e implementación de estructuras de datos.
- Uso imperativo de estructuras de datos. Iteración en listas y árboles.
- Modelo de memoria imperativo: stack/heap, asignación de memoria. Punteros. Variables por referencia.
- Listas encadenadas y sus variantes. Árboles implementados con punteros. Binary heaps implementados con arrays.

- 
- Hashing. Análisis de eficiencia e implementación.
  - Algoritmos de ordenamiento. Clasificación e implementación.
  - Nociones básicas de algoritmos sobre grafos.

---

### 3.4.7. Programación con Objetos II

#### Objetivos

Que el estudiante:

- Adquiera ideas y técnicas que le permitan aprovechar distintas características de la orientación a objetos (como ser comportamiento, responsabilidad, polimorfismo, uso de patrones de diseño) para dar soluciones adecuadas a los desafíos planteados, que apliquen ya sea al proceso de diseño (cómo atacar un dominio) ya sea al software que se está diseñando (en particular, respecto de los conceptos de acoplamiento y cohesión).
- En particular, adquiera cierta fluidez en el manejo del concepto de tipo como herramienta para estructurar un programa: qué tipo se espera de un valor que se recibe, qué tipos puede ofrecer un valor que se define.
- Comprenda las consecuencias de trabajar en un entorno con chequeo estático de tipos.
- Comprenda la idea de diseño de software.
- Comprenda los desafíos que aparecen al construir aplicaciones más grandes, en las que el dominio a modelar es más complejo, y cuyo desarrollo lleva más tiempo.
- Conozca formas de organizar los componentes de un programa en unidades mayores a los bloques de construcción básicos; p.ej. utilizando módulos, subsistemas y/o packages.
- Comprenda la necesidad de controlar el acoplamiento entre los componentes que forman un sistema de software.
- Se enfrente a la necesidad de adaptar un producto de software en construcción o construido a cambios en los requerimientos que afectan al diseño definido.
- Haga una experiencia de uso intensivo de un entorno de desarrollo integrado del estilo de los usados habitualmente en la industria.

#### Contenidos mínimos

- Aproximación al diseño de software: en qué dimensiones puede crecer un proyecto de software, problemáticas que devienen de este crecimiento, necesidad de pensar en la organización de un sistema como elementos relacionados, pensando en la funcionalidad de cada elemento y de qué relaciones se establecen. Noción de decisión de diseño, el diseño es un proceso de toma de decisiones.
- Conceptos de acoplamiento y cohesión. Problemas que derivan de un grado de acoplamiento inadecuado.
- Vinculación entre las ideas básicas de diseño y el paradigma de objetos: objetos como elementos, conocimiento como relación, responsabilidades como funcionalidad de cada elemento, tipo y polimorfismo para comprender que ciertos elementos son intercambiables a efectos de un diseño.
- Características deseadas en un diseño de objetos: no repetición de implementación de lógica, capacidad de separar entre grupos de objetos cohesivos con responsabilidades aplicables al grupo.
- Patrones de diseño: idea de patrón, consecuencias del uso de algunos patrones respecto de las características del diseño y de las cualidades pretendidas del producto.

- 
- Nociones sobre proceso de diseño: foco en las responsabilidades, pensar los problemas desde las características básicas del paradigma, pertinencia de iterar entre diseño en papel, codificación y test, relevancia de los diagramas de objetos y de clases.
  - Metaprogramación, características reflexivas de un lenguaje de programación.
  - Uso de un entorno integrado de software del estilo de los utilizados ampliamente en la industria, funcionalidades que provee, aprovechamiento de sus facilidades.
  - Notación UML de los diagramas de clases, de objetos y de secuencia.

#### **Contenidos sugeridos**

- Noción de cualidades pretendidas en un producto de software, en particular las más ligadas con el desarrollo de modelos de negocio: modificabilidad, extensibilidad, robustez, performance, escalabilidad, etc..
- Profundización del trabajo sobre manejo de errores.

### **3.5. Objetivos y contenidos mínimos de las materias del núcleo avanzado obligatorio**

Se consignan, para cada materia del núcleo avanzado obligatorio, sus objetivos y contenidos mínimos. Asimismo, para algunas asignaturas se indican contenidos sugeridos que complementan adecuadamente los contenidos consignados como mínimos.

---

### 3.5.1. Redes de Computadoras

#### Objetivos

Que el estudiante:

- Se familiarice con el concepto de un modelo en capas.
- Comprenda la aplicación del concepto de protocolo en varios niveles.
- Conozca con cierta profundidad arquitectura, protocolos, normas y aplicaciones típicas de las principales redes de datos, en particular las redes de área local y la Internet.

#### Contenidos mínimos

- Concepto de red de computadoras, redes y comunicación.
- Modelos en capas, modelo OSI, modelo de la Internet.
- Conceptos de protocolo y de servicio.
- Nivel físico: dispositivos, cableado estructurado.
- Nivel de enlace: concepto de enlace, tramas, puentes, enlaces inalámbricos.
- Nivel de red: concepto de ruteo, topologías, algoritmos de ruteo, protocolos IP, resolución de direcciones.
- Nivel de transporte: funciones, protocolos UDP y TCP, multiplexación, concepto de socket, control de congestión.
- Modelo general de Internet: integración de niveles y protocolos, servicios de red (http, dhcp, dns, smtp, etc.), su utilización en el funcionamiento de la Web.
- Estándares utilizados en Internet, concepto de RFC.
- Concepto e implementación de las VPN.
- Administración de redes: servicios, firewalls.
- Sistemas cliente/servidor.

---

### 3.5.2. Sistemas Operativos

#### Objetivos

Que el estudiante:

- Conozca las funciones de un sistema operativo, las características de los recursos que administra y las diferentes políticas de administración.
- Conozca los conceptos de proceso y thread, la necesidad de planificar y controlar su ejecución, y tenga nociones de políticas de planificación.
- Se familiarice con los entornos operativos del estilo Unix, el uso de interfaz de línea de comandos, y adquiera conocimientos sobre shell scripting.

#### Contenidos mínimos

- Introducción a los sistemas operativos: función de abstracción del hardware; organización, estructura y servicios de los SO. Tipos de sistemas (Sistemas batch / Multiprogramación / Sistemas de tiempo real / Sistemas distribuidos / Sistemas paralelos).
- Procesos y threads: Conceptos de proceso, thread y planificación. Comunicación y cooperación entre procesos. Deadlocks.
- Planificación: Algoritmos, criterios. Multiprocesamiento.
- Manejo de memoria: Espacio lógico vs físico, swapping, alocaión contigua, paginación, segmentación.
- Memoria virtual : Paginación bajo demanda, algoritmos de reemplazo de página, thrashing.
- Sistemas de archivos: Manejo de archivos, manejo de directorios.
- Protección: objetivos, dominio de protección, matriz de acceso y sus implementaciones.
- Prácticas, trabajos incluyendo uso de shell scripting e instalaciones en distintos sistemas operativos, en particular del estilo Unix: GNU/Linux, etc..

---

### 3.5.3. Programación Concurrente

#### Objetivos

Que el estudiante:

- Conozca los modelos más utilizados y las técnicas y conceptos básicos asociados a la programación concurrente.
- Comprenda las dificultades asociadas a la interacción de componentes en un sistema concurrente y conozca los recursos del que dispone el programador para mitigarlos.
- Pueda analizar programas concurrentes.
- Pueda desarrollar aplicaciones concurrentes sencillas.

#### Contenidos mínimos

- Los por qué de la concurrencia. Concurrencia vs paralelismo.
- Modelo de memoria compartida, atomicidad e independencia.
- Secciones críticas, locks y barriers, semáforos, monitores y condition variables, Rendezvous.
- Problemas de la concurrencia: Starvation, Deadlocks, Liveness y Progress, Safety, Race conditions, Fairness.
- Modelo de pasaje de mensajes: Comunicación sincrónica vs comunicación asincrónica, Modelo de transacciones.
- Modelos de interacción: Cliente/Servidor, Productor/Consumidor.
- Aplicación de los conceptos estudiados en lenguajes de programación concretos, mecanismos de sincronización.

---

### 3.5.4. Matemática II

#### Objetivos

Son objetivos generales:

- Profundizar la capacidad de abstraer estructuras a partir de casos concretos, y de razonar acerca de estructuras abstractas.
- Proporcionar algunas bases matemáticas imprescindibles, montadas sobre las que se ven en Matemática I, en concordancia con el área de Programación, de modo de contribuir a la formación integral del Técnico Universitario en Programación Informática.
- Fortalecer algunos procesos típicos del pensamiento matemático: conjeturar, inducir, deducir, probar, generalizar, particularizar, modelar y representar algunos conceptos. En particular, ser capaz de articular algunos razonamientos probabilísticos.

Son objetivos específicos que el estudiante logre:

- Incorporar nociones básicas de probabilidad de modo de contribuir a la comprensión del uso de estos conceptos en el marco de áreas de aplicación tales como modelado de texto y datos web, tráfico de red, análisis de algoritmos, data mining, etc.
- Utilizar el lenguaje matricial y su operatoria para la resolución de sistemas de ecuaciones lineales fortaleciendo la comprensión de su uso en áreas como algoritmos, computación gráfica, geometría computacional, etc.
- Reconocer las estructuras fundamentales del álgebra, interpretarlas y construir modelos de las mismas contribuyendo a establecer paralelos con estructuras semejantes que se pueden atribuir a conjuntos de cadenas, lenguajes y programas.

#### Contenidos mínimos

- Aritmética entera y modular.
- Introducción a las probabilidades, distribuciones de probabilidad discreta.
- Matrices y sistemas de ecuaciones lineales.
- Estructuras algebraicas: monoides, semigrupos y grupos.



---

### 3.5.5. Elementos de Ingeniería de Software

#### Objetivos

Que el estudiante:

- Entienda que para llevar a cabo un proyecto de desarrollo de software hace falta llevar a cabo varias actividades además de programar, y tenga una noción de cuáles son estas actividades y las técnicas asociadas a cada una.
- Conozca el concepto de metodología como definición de las actividades que involucra el desarrollo de software, su articulación y los roles que ocupan las personas que participan.
- Conozca los conceptos principales asociados a metodologías ágiles y estructuradas, las actividades y roles que involucran, y algunas similitudes y diferencias entre ambos enfoques.
- Pueda interpretar requerimientos funcionales y no funcionales, y tenga noción de las actividades asociadas a tareas de programación necesarias para lograr la concreción y verificación de los mismos.
- Comprenda la relevancia de los distintos tipos de testing existentes y el alcance de cada uno de ellos e identifique cuáles son los más relacionados con las actividades de un programador.
- Tenga una pequeña experiencia práctica aplicando las actividades y metodologías que se describen en la materia.

#### Contenidos mínimos

- Metodologías ágiles: actividades, productos, formas de articulación, roles. Ejemplos: Scrum.
- Metodologías estructuradas: actividades, productos, formas de articulación, roles. Ejemplos: UP.
- Debate sobre similitudes y diferencias entre metodologías ágiles y estructuradas.
- Concepto de ciclo de vida, relación con distintas metodologías.
- Métricas: qué son, qué miden, para qué sirven, cuándo sirven. Ejemplos de métricas asociadas a desarrollo de software en general y actividades de programación en particular.
- Estimación de esfuerzos: relevancia de la experiencia previa para estimar, heurísticas utilizadas. Pertinencia de estimaciones relativas. Técnicas de estimación asociadas a metodologías ágiles.
- Conceptos de requerimiento funcional y no funcional, pertinencia de definiciones comprensibles y adecuadas.
- Comprensión de requerimientos funcionales, detección de inconsistencias. Implementación en código de requerimientos funcionales, verificación de que el código construido cumple los requerimientos.
- Problemas asociados a requerimientos no funcionales: pertinencia de definiciones medibles, nociones sobre técnicas de verificación, posibilidad de garantizarlos por construcción.

- 
- Nociones sobre distintos tipos de testing: de unidad, funcional, de sistema, de stress, de carga. Cualidades deseadas y técnicas para lograrlas: regresión, automatización, independencia. Ejemplos concretos de test de unidad y de test funcional. Noción de coverage.
  - Prácticas asociadas a extreme programming: peer programming, relevancia de tests automáticos, integración continua, interacción de las actividades de coding y refactor. Noción de TDD.
  - Nociones de riesgo y plan de contingencia.

#### **Contenidos sugeridos**

- Nociones básicas sobre Software Configuration Management: versionado, generación de productos instalables, despliegue y redespliegue. Problemas asociados al despliegue de aplicaciones.

---

### 3.5.6. Construcción de Interfaces de Usuario

#### Objetivos

Que el estudiante:

- Conozca la problemática específica de la construcción de interfaces de usuario, comprendiendo sus requerimientos y restricciones funcionales, tecnológicos, arquitecturales y de usabilidad.
- Comprenda los distintos componentes que se pueden utilizar para construir una interfaz de usuario, las formas de organizar esos componentes, los patrones de diseño más comunes asociados a esta problemática.
- Conozca, experimente y pueda comparar distintas alternativas para la descripción de interfaces de usuario y su vinculación con el modelo de dominio subyacente, y comprenda la pertinencia tanto de permitir la evolución independiente del modelo de dominio y de la interfaz con el usuario como de acotar el impacto de la problemática tecnológica de las interfaces de usuario en los componentes no vinculados directamente con la tecnología.
- Sea capaz de utilizar distintas herramientas de programación, diseño y metodológicas que conoce previamente en el dominio específico de la presentación, en particular diseño orientado a objetos, unit testing y manejo de errores.
- Comprenda la problemática básica de las interfaces remotas.
- Conozca diferentes opciones tecnológicas utilizadas en la industria y adquiera algunos criterios que lo ayuden a discernir entre distintas alternativas permitiéndole elegir las más adecuadas a las condiciones de un proyecto de desarrollo específico.
- Cuente con elementos para aprender una tecnología vinculada a la construcción de interfaces de usuario desconocida por él/ella e incluirla en el marco de los conocimientos vistos en la materia.
- Sea capaz de construir aplicaciones sencillas aplicando los conceptos vistos en la materia y utilizando distintas tecnologías.
- Reconozca y adquiera algunas buenas prácticas de aplicación particular al desarrollo de la interfaz de usuario de una aplicación.

#### Contenidos mínimos

- Variantes en arquitecturas de sistema respecto de la interfaz de usuario: aplicación centralizada, cliente-servidor o distribuida; ejecución en un cliente de aplicación (browser, flash, otros) o mediante un programa específico; concepto de RIA.
- Arquitecturas web, protocolos y tecnologías asociados.
- Modelos de interacción de la interfaz de usuario con su entorno: interfaces orientadas a eventos, pedido-respuesta, basadas en continuations. Aplicaciones *client-initiative* y *application-initiative*.
- Componentes gráficos usuales en interfaces de usuario. Diferentes estrategias para describir una vista, sus componentes y la distribución espacial de los mismos: HTML estático, CSS, generación programática de HTML, server pages, templates, descripción basada en componentes, descripciones declarativas. Problemas característicos de cada estrategia; herramientas que las soportan.

- 
- Vinculación entre la interfaz de usuario y el modelo de dominio subyacente. Problemática asociada a transformaciones, validaciones, manejo de errores, excepciones, transacciones e identidad. Distintos enfoques: generación automática de la interfaz de usuario a partir del modelo, vínculos explícitos entre elementos de interfaz de usuario y de modelo, DAOs, servicios.
  - Adaptaciones de un modelo de dominio a las necesidades de dinamismo, navegación y distintos niveles de discriminación/agregación de la interfaz de usuario. Objetos de nivel de aplicación, casos de uso, concepto de modelo de la vista. Patrones de interacción, mvc.
  - Análisis de tecnologías de presentación de acuerdo a los conceptos presentados en esta materia; evaluación de características, selección de opciones tecnológicas teniendo en cuenta el proyecto de desarrollo a realizar. Nociones sobre desarrollo propio de complementos a tecnologías desarrolladas por otros.
  - Impacto de la distribución de aplicaciones en la interfaz de usuario, comunicación sincrónica y asincrónica.
  - Navegación y manejo del estado conversacional. REST, estado en sesión.
  - Nociones de usabilidad: concepto, pertinencia, conveniencia de definir y mantener standards.

#### **Contenidos sugeridos**

- Aspectos de seguridad vinculados a la interfaz de usuario, en particular control de accesos a elementos de la interfaz.
- Análisis de performance de una interfaz de usuario, pruebas de stress y carga.
- Testeo unitario de interfaces de usuario: desafíos, opciones, posible impacto en el diseño de software de las interfaces.

---

### 3.5.7. Estrategias de Persistencia

#### Objetivos

Que el estudiante:

- Conozca distintos mecanismos de persistencia, en particular persistencia en archivos y persistencia en bases de datos relacionales u orientadas a objetos; haga experiencia práctica con ellos y tenga elementos para realizar comparaciones.
- Entienda las ideas básicas para poder interactuar con un mecanismo de persistencia desde un programa externo, las problemáticas asociadas de transformación entre estructuras de persistencia y estructuras del programa, y la pertinencia de acotar el ámbito en el cual el mecanismo de persistencia toma preeminencia sobre las abstracciones encontradas en el programa.
- Comprenda la problemática específica de la persistencia de objetos y de su implementación en distintos mecanismos de persistencia, en particular la impedancia objetos/relacional.
- Entienda los problemas de concurrencia específicos que derivan del acceso a un mecanismo de persistencia, y tenga experiencia práctica en algunas técnicas para tratarlos, analizando tanto los mecanismos de concurrencia como su inserción en programas externos.
- Conozca algunas técnicas que permitan trabajar sobre la performance de mecanismos de concurrencia y entienda en qué casos es adecuado evaluarlas y aplicarlas, analizando tanto los mecanismos de concurrencia como su inserción en programas externos. En particular, en relación con bases de mediano o gran volumen de información.
- Entienda y pueda manejar en la práctica conceptos relacionados con la seguridad en el acceso a un mecanismo de persistencia.

#### Contenidos mínimos

- Nociones sobre los problemas que derivan del acceso concurrente a una base de datos. Algunas estrategias para mitigarlos, en particular lockeo y manejo adecuado de transacciones.
- Nociones sobre la problemática de performance en el acceso a una base de datos, relación con la escala, otros factores que influyen. Estrategias de acceso a los datos ante una consulta, concepto de índice.
- Conceptos de usuario y permiso en una base de datos, esquemas típicos de definición de usuarios y permisos.
- Bases de objetos: concepto, panorama, experimentación práctica, comparación con bases de datos relacionales.
- Bases de datos distribuidas para grandes volúmenes de datos, acceso a datos como un servicio, herramientas de programación asociadas.
- Interacción entre un programa y un mecanismo de persistencia: nociones básicas, problemáticas generales.
- Mecanismos de acceso y recuperación de objetos persistidos en bases de datos relacionales: mecanismos de recuperación de objetos (uso de lenguajes de consulta relacionales, lenguajes de consulta orientados a objetos, interfaz en objetos orientada al acceso, interfaces en términos del modelo de dominio). Actualización del estado persistente: reachability, cascada.

- 
- ORM, conceptos básicos, alcances, cuestiones que resuelven, enfoque que toma respecto de la transformación de objetos. Problemas de mapeo: herencia, relaciones n-m, estrategias no standard.
  - Transacciones a nivel aplicación, transacciones de negocio, reflejo de la transaccionalidad al acceder a un mecanismo de persistencia, concepto de unit of work.
  - Reflejo de cuestiones de performance y concurrencia al acceder a un mecanismo de persistencia desde un programa, lazyness, cache, versionado, lockeo optimista y pesimista.

---

### 3.5.8. Programación Funcional

#### Objetivos

Son objetivos generales que el estudiante:

- Amplíe su comprensión sobre el concepto de programación como actividad rigurosa de abstracción, incluyendo el paradigma de programación funcional.
- Explore y utilice con seguridad los principales conceptos del paradigma funcional.
- Conozca las principales técnicas formales de desarrollo de programas, comprendiendo su importancia y utilidad como medio de garantizar la corrección del software producido.

Es objetivo específico que el estudiante:

- Comprenda, maneje y se familiarice con conceptos fundamentales de la programación y su importancia en la tarea de programar:
  - Abstracción mediante funciones (como elementos que transforman información),
  - Noción de funciones de alto orden y su utilidad, currificación, esquemas de programas,
  - Inducción (y las herramientas asociadas: inducción estructural y recursión),
  - Nociones de sistemas de tipos y su utilidad práctica,
- Sea capaz de utilizar dichas nociones para la confección de programas sencillos en un lenguaje funcional.
- Sea capaz de demostrar propiedades sencillas de programas funcionales utilizando inducción estructural.
- Sea capaz de aplicar técnicas de transformación de programas en casos particulares.

#### Contenidos mínimos

- Nociones generales del paradigma funcional
  - Valores y expresiones. Las funciones como valores. Mecanismos de definición de expresiones y valores. Ecuaciones orientadas para definir funciones. Sintaxis.
  - Sistema de Tipos Hindley-Milner. Tipos básicos. Constructores de tipos. Polimorfismo. Sintaxis para valores de cada tipo (caracteres, tuplas, listas, strings, funciones).
  - Funciones de alto orden. Currificación.
- Inducción/Recursión
  - Definición inductiva de conjuntos.
  - Definición recursiva de funciones sobre esos conjuntos.
  - Demostraciones inductivas sobre dichas funciones. Inducción estructural.
  - Ejemplos: programas, expresiones aritméticas, listas.
- Listas
  - Listas como tipo inductivo. Funciones básicas sobre listas (append, head, tail, take, drop, reverse, sort, elem, etc.).

- 
- Funciones de alto orden sobre listas. Patrón de recorrido: map. Patrón de selección: filter. Patrón de recursión: foldr.
  - Demostración de propiedades sobre listas y funciones sobre listas.
  - Sistemas de Tipos.
    - Nociones básicas. Sistemas de tipado fuerte. Ventajas y limitaciones de los lenguajes de programación con tipos.
    - Lenguaje de tipos. Asignación de tipos a expresiones. Propiedades interesantes de esta asignación. Algoritmo de inferencia.
    - Mecanismos de definición de tipos nuevos y de funciones sobre ellos. Tipos algebraicos recursivos. Ejemplos: enumeraciones, listas, árboles binarios, árboles generales.
  - Transformación de Programas.
    - Motivación. Obtención de programas a partir de especificaciones. Mejoramiento de eficiencia, con corrección por construcción.
    - Técnicas particulares de transformación: tupling, eliminación de recursión, fusión.



---

### 3.5.9. Desarrollo de Aplicaciones

#### Objetivos

Que el estudiante:

- Lleve a cabo la construcción completa de una aplicación mediana, partiendo desde una versión preliminar de los requerimientos y llegando a una aplicación que pueda ponerse en producción.
- Aplique en una experiencia concreta de desarrollo los conocimientos sobre ciclo de desarrollo y metodologías adquiridos en Elementos de Ingeniería de Software.
- Comprenda los beneficios de respetar buenos principios y prácticas adquiridos en materias anteriores en un proyecto de desarrollo de software, mediante la experiencia concreta en la utilización e integración de los mismos.
- Tenga una noción de qué es una arquitectura de software, y qué consecuencias trae trabajar con una determinada arquitectura en el desarrollo.
- Tenga un conocimiento razonable acerca de herramientas tecnológicas, metodológicas y de diseño de software adecuadas para el desarrollo grupal de proyectos de software.
- Tenga nociones acerca del análisis de eventuales problemas de performance, la pertinencia de abordarlos en los momentos y en la medida adecuados, y la relevancia de una visión global al respecto incluyendo tanto cuestiones de configuración como cuestiones de código.
- Tenga nociones acerca de los desafíos y tareas inherentes al mantenimiento de una aplicación posterior a su primer puesta en producción.

#### Contenidos mínimos

- Herramientas metodológicas y conceptuales para trabajo en grupo. División de tareas planeando reunión de los resultados. Aprovechamiento de conceptos de objetos, citamos como posibles ejemplos: interfaces como forma de coordinar la tarea de distintas personas o grupos, *mock objects* para simular los objetos de otros grupos, etc..
- Herramientas tecnológicas para trabajo en grupo: repositorios de código y de biblioteca, automatización de procesos involucrados en el desarrollo y despliegue.
- Aplicación concreta de las ideas de desarrollo iterativo.
- Nociones de arquitectura de software: qué es una decisión de arquitectura, impacto en el desarrollo de trabajar dentro de cierta arquitectura. Diversidad de variantes en arquitecturas de software.
- Arquitectura de sistema. Variantes: stand-alone, servidor Web dinámico, servidor de aplicaciones, cliente-servidor. Soporte de ejecución concurrente. Implicancias en el diseño de software y en las tareas habituales de desarrollo, pertinencia de prácticas que agilicen el desarrollo bajo distintas arquitecturas de sistemas.
- Problemas de performance: instancias adecuadas para su análisis, posibilidad de problemas asociados a configuraciones defectuosas.
- Despliegue de una aplicación en distintos entornos: tareas que conlleva.

#### Contenidos sugeridos

- 
- Nociones básicas de manejo de configuraciones: versionado, generación de una versión instalable, branches para desarrollo y para corrección de errores.
  - Nociones de tareas que devienen de un cambio de versión: conversiones de datos, instalación de nuevas versiones de bibliotecas, etc..
  - Técnicas para organización de las tareas individuales de un desarrollador de software.
  - Experiencia con ambientes de integración continua.
  - Concepto de servicio, cómo se plasma en una pieza de software que provee un servicio, cómo se utiliza un servicio.

---

### **3.5.10. Laboratorio de Sistemas Operativos y Redes**

#### **Objetivos**

Que el estudiante:

- Conozca los requerimientos de infraestructura de hardware, red y entorno operativo necesarios tanto para desarrollar como para instalar distintos productos de software.
- Pueda instalar, configurar, operar y mantener un conjunto de servicios de soporte para el desarrollo de un sistema.
- Integre los conocimientos adquiridos en las otras materias del área (Organización de Computadoras, Redes de Computadoras y Sistemas Operativos) para ganar profundidad de comprensión respecto de tareas operativas.

#### **Contenidos mínimos**

- Instalación, configuración y operación de distintos servicios relacionados con Internet: servidores de aplicaciones, servidor y cliente de mail, servidor y cliente FTP, firewalls, etc..
- Servicios de directorio, servidores LDAP, uso desde aplicaciones.
- Gestión de usuarios y control de accesos en un entorno operativo, impacto en la instalación de aplicaciones, posibilidad de compartir recursos.
- Sistemas de backup automatizados, políticas de criticidad.
- Instalación, configuración y operación de repositorios de código.
- Monitoreo de redes, protocolo SNMP.

---

### **3.6. Contenidos mínimos de los niveles de idioma inglés**

Se consignan los contenidos mínimos de los niveles de inglés que cada estudiante debe acreditar.

#### **3.6.1. Inglés I**

- Comunicación oral y escrita sobre la base de temáticas profesionales preferentemente, comprensión de textos y producción de textos orales y escritos.
- Formación del vocabulario técnico.
- Práctica intensiva de traducción con referencia especial a obras profesionales.

#### **3.6.2. Inglés II**

- Presentación de textos profesionales considerando al inglés como idioma vivo, como útil de trabajo; para ello se utilizarán textos tomados de las asignaturas de la carrera y de la red Internet. El contenido gramatical se restringe a las esenciales y típicas en un contexto científico, en particular no recorre todo el abanico de estructuras posibles de inglés general.
- Formación del vocabulario técnico, de traducción con referencia especial a obras profesionales. Comprensión de discursos orales vinculados con la vida profesional.

---

### 3.7. Contenidos mínimos de las materias del núcleo complementario

Se consignan los contenidos mínimos de las materias del núcleo complementario.

#### 3.7.1. Seguridad Informática

- Conceptos de seguridad informática: ataques, amenazas, servicios, herramientas, estándares.
- Firma Digital.
- Nociones básicas de criptografía, criterios de calidad, algoritmos.
- Seguridad en redes: protocolos asociados a la seguridad, autenticación de equipos en redes de computadora.

#### 3.7.2. Bases de Datos II

- Cuestiones de eficiencia en el acceso a bases de datos, entre otras: transformación de consultas, *hints* al motor, trabajo sobre índices.
- Configuraciones de nivel físico en un motor de base de datos relacional, p.ej. tablespaces y replicación.
- Tipos de datos no-standard en bases de datos, como ser blobs o XML.
- Implementación física de bases de datos relacionales, en particular: manejo eficiente de archivos, implementación de índices usando árboles B y variantes.
- Conceptos básicos de Data Mining y Data Warehousing.

#### 3.7.3. Participación y Gestión en Proyectos de Software Libre

- Cibercultura y cultura hacker. Nuevos modos de relacionarse en internet: cultura abierta, distribuida, libre, producción colaborativa en red.
- Idea de software libre, movimiento de software libre, principios, principales productos y logros.
- Participación en proyectos de software libre: fuentes de información, formas que puede asumir la participación.
- Creación de proyectos de software libre: de la idea a la formulación
- El sitio de la comunidad del proyecto: forjas de software libre y otros espacios de trabajo colaborativo.
- Herramientas para el desarrollo de un proyecto de software libre, en particular: herramientas de comunicación del proyecto, de análisis y diseño y desarrollo de aplicaciones, de gestión de código y control de versiones, de gestión de la documentación
- Gestión de la admisión de contribuciones, requerimientos, errores y parches.
- Etiqueta en la comunicaciones electrónicas en el marco de los proyectos.
- Motivaciones de los desarrolladores y de los grupos de software libre.
- Roles usados más frecuentemente, mecanismos de decisión dentro del proyecto.

- 
- Bifurcaciones de proyectos, conexiones entre proyectos, cierre de proyectos.
  - Licencias para obras intelectuales, en particular para software y para su documentación técnica asociada. Licencias de software libre. BSD. GNU. Mozilla.
  - Experiencia concreta de participación en al menos un proyecto existente

#### **3.7.4. Introducción a las Arquitecturas de Software**

- Conceptos básicos: qué es una arquitectura de software, principios de arquitectura, objetivos de la definición de una arquitectura.
- Relación entre arquitectura de software y requisitos no funcionales.
- Rol de las arquitecturas en la obtención de parámetros de calidad de software.
- Impacto de la elección de una arquitectura en el desarrollo de software: necesidad de respetar estándares, comprensión de los aspectos que resuelve una arquitectura determinada.

#### **3.7.5. Programación con Objetos III**

- Metaprogramación: concepto, herramientas, posibilidades que brinda, uso en distintas herramientas genéricas, posibilidad de análisis estático.
- Programación aprovechando aspectos, conceptos, relaciones con la programación con objetos, implementaciones.
- Posibilidad de combinar características de los paradigmas funcional y de objetos: manejo de la estrategia de evaluación, objetos que representan funciones.
- Alternativas dentro del paradigma: herencia múltiple, mixins, traits, prototipos.
- Desarrollo de extensiones a entornos de programación.

#### **3.7.6. Introducción a la Bioinformática**

- Conceptos básicos de la genética molecular: leyes de la herencia, genética de poblaciones, genética evolutiva, replicación del ADN, mutación y reparación.
- Acceso remoto a bancos de datos, bancos genéticos.
- Análisis de secuencias biológicas, algoritmos asociados.
- Homologías secuenciales y estructurales.

#### **3.7.7. Políticas Públicas en la Sociedad de la Información y la Era Digital**

- Estado y políticas públicas.
- Cultura abierta, distribuida, libre, producción colaborativa en red.
- Derechos en la sociedad de la información.
- Diferentes iniciativas públicas referentes a los estándares abiertos y al software libre.
- Diversidad e identidad culturales, diversidad lingüística y contenidos locales.
- Sociedad de la información y el conocimiento.
- Proyectos de infraestructura y accesibilidad TICs.

- 
- Acceso y usos: de la red, de los contenidos.
  - Datos abiertos, gobierno electrónico, gobierno abierto, democracia electrónica. Planteo y eventual desarrollo de algún software relacionado con esta temática.
  - Neutralidad en la red.

### **3.7.8. Sistemas de Información Geográfica**

- Introducción a los Sistemas de Información Geográfica (GIS): objetivos, principales tecnologías utilizadas.
- Posicionamiento: coordenadas, sistemas de referencia, proyecciones, datums, precisión.
- Modelos de datos: vectorial, raster, interpolación, implementaciones de formatos (SHP, GeoTIFF, KML, otros).
- Bases de datos espaciales: tipos de datos, consultas, índices.
- Servidores de Mapas: protocolos, en particular WMS y WFS; tecnologías. Clientes de Mapas: protocolos y tecnologías.
- Sistemas de Información Geográfica de Escritorio.
- Implementación de GIS con tecnologías OpenSource: servidor de Bases de Datos, servidor de Mapas, clientes Desktop y Web.

### **3.7.9. Herramientas Declarativas en Programación**

- Enfoques imperativo y declarativo de la programación informática, sus diferencias, consecuencias de adoptar un enfoque declarativo.
- Bases del paradigma de programación lógico: describir un programa definiendo relaciones, concepto de cláusula, inversibilidad, principio de universo cerrado.
- Posibilidad de utilizar conceptos de la programación lógica en entornos de objetos o procedurales, programación de motores de reglas.
- Posibilidad de combinar características de los paradigmas funcional y de objetos: manejo de la estrategia de evaluación, objetos que representan funciones.
- Aplicación de un enfoque declarativo en la construcción de interfaces de usuario: separación de los detalles de visualización, generación de la interfaz a partir de un modelo de objetos a renderizar.
- Lenguajes de dominio específico (DSL): concepto, separación entre especificación de dominio y código común, modelo semántico, experimentación con herramientas concretas.

### **3.7.10. Introducción al Desarrollo de Videojuegos**

- Panorama de la historia y estado corriente de la industria de videojuegos.
- Diversidad de videojuegos, géneros mejor establecidos.
- Concepto de game design, relevancia del relato al pensar el concepto de un juego.

- 
- Aspectos generales en la concepción de videojuegos: estilos visuales, relevancia de la experiencia interactiva del usuario, necesidad de testeo subjetivo, pertinencia de conceptos de modelado físico.
  - Cuestiones de arquitectura de software y hardware pertinentes para el dominio de videojuegos: game loop, arquitecturas P2P o cliente-servidor para juegos multiplayer, necesidad de sincronización de estados en distintas terminales.
  - El proceso de desarrollo de videojuegos, pertinencia de aplicar conceptos ágiles.
  - Características y bondades del modelado de un juego utilizando los conceptos de la programación con objetos: modelado del dominio en función del game design, modelado del comportamiento aprovechando el polimorfismo, modelado del flujo interactivo usando estados.
  - Relevancia del procesamiento de eventos en varios géneros de juegos.
  - Cuestiones ligadas al tratamiento de gráficos: uso extensivo de bibliotecas gráficas y buenas prácticas para su integración en una arquitectura de software, sprites, meshes, frustum, cálculo de colisiones.

#### **3.7.11. Derechos de Autor y Derecho de Copia en la Era Digital**

- La arquitectura jurídico-política del derecho de autor y derecho de copia.
- El derecho de autor y derecho de copia y su relación con el cambio tecnológico.
- Los derechos personales/morales y los derechos patrimoniales de autor.
- Propiedad intelectual. Patentes, marcas y logotipos.
- Las relaciones laborales y las presunciones legales sobre la titularidad de las obras.
- Las obras intelectuales, sus formas de expresión en soportes y la duplicidad de sus regulaciones
- El derecho de copia como construcción jurídico-política.
- Las licencias abiertas / libres, recíprocas/permisivas/mixtas, el concepto del copyleft, el sistema de licencias abiertas / libre de Creative Commons, otras licencias.
- El software libre, el software de fuente abierta (open source), software privativo y software privado o no publicado.
- Dominio público.

#### **3.7.12. Seminarios**

Se trata de cursos sobre temáticas específicas correspondientes a las características dinámicas del ámbito de la programación, relacionadas con

- temas avanzados de programación.
- dominios o tipos específicos de proyectos de software.
- herramientas que cuenten con un real interés para la complementación de la formación de los estudiantes.

#### **3.7.13. Seminarios sobre Herramientas o Técnicas Puntuales**

Se trata de cursos que brindan al estudiante la posibilidad de conocer y experimentar con herramientas o técnicas de programación de especial interés para determinados dominios de aplicación.



---

## 4. Cuadro resumen

Materia	Horas semanales	Carga horaria total	Créditos
<b>Núcleo básico obligatorio</b> El alumno deberá aprobar todas las asignaturas de este núcleo.			
Introducción a la Programación	8	144	16
Organización de Computadoras	6	108	12
Matemática I	8	144	16
Programación con Objetos I	8	144	16
Bases de Datos	6	108	12
Estructuras de Datos	8	144	16
Programación con Objetos II	6	108	12
<b>Núcleo avanzado obligatorio</b> El alumno deberá acreditar la aprobación del 70 % de los créditos del núcleo básico obligatorio para comenzar a cursar materias de este núcleo. El alumno deberá aprobar todas las asignaturas de este núcleo.			
Redes de Computadoras	6	108	12
Sistemas Operativos	6	108	12
Programación Concurrente	4	72	8
Matemática II	4	72	8
Elementos de Ingeniería de Software	6	108	12
Construcción de Interfaces de Usuario	6	108	12
Estrategias de Persistencia	6	108	12
Programación Funcional	4	72	8
Desarrollo de Aplicaciones	6	108	12
Laboratorio de Sistemas Operativos y Redes	4	72	8

---

Materia	Horas semanales	Carga horaria total	Créditos
<b>Núcleo complementario</b> El alumno deberá acreditar la aprobación del 100 % de los créditos del núcleo básico obligatorio para comenzar a cursar materias de este núcleo. El alumno deberá aprobar asignaturas de este núcleo por un mínimo de 24 créditos, que se corresponden con 216 horas de cursada.			
Seguridad Informática	4	72	8
Bases de Datos II	4	72	8
Participación y Gestión en Proyectos de Software Libre	4	72	8
Introducción a las Arquitecturas de Software	4	72	8
Programación con Objetos III	4	72	8
Introducción a la Bioinformática	4	72	8
Políticas Públicas en la Sociedad de la Información y la Era Digital	4	72	8
Sistemas de Información Geográfica	4	72	8
Herramientas Declarativas en Programación	4	72	8
Introducción al Desarrollo de Videojuegos	4	72	8
Derechos de Autor y Derecho de Copia en la Era Digital	4	32	8
Seminarios	4	72	8
Seminarios sobre Herramientas o Técnicas Puntuales		32	4

## Asignación horaria total

Es de 2260 horas de acuerdo al detalle del cuadro siguiente

Núcleo	Carga horaria total mínima	Créditos
Básico obligatorio	900	102
Avanzado obligatorio	936	104
Complementario	216	24
Trabajo de de inserción profesional	100	10
Otros requisitos (niveles de inglés)	108	0
<b>Totales</b>	<b>2260</b>	<b>240</b>