## PLANTILLA PARA EL PLAN SINTÉTICO DE UN ESPACIO PEDAGÓGICO

|  |  |  |  |
| --- | --- | --- | --- |
| **Espacio Pedagógico:** Lenguajes de programación | | | |
| **Código:** | | | |
| **Unidad Académica Responsable:** Vicerectoría Académica | | | |
| **Carrera según grado:** Profesor de informática Con orientación en Robótica o Diseño Gráfico en el grado de Licenciatura. | | | |
| **Requisitos (código, nombre y unidades valorativas):**  Estructuras de datos y algoritmos | | | |
| **Distribución de la actividad académica del espacio pedagógico:** | | | |
| **Total de unidades valorativas:** 5  **Teóricas:** 3  **Práctica:** 2 | **Número de semanas:** 13  **Horas por semana Teóricas:** 45  **Horas por semana Prácticas:** 30 | Horas de clase frente al profesor en el período académico y durante la **semana:**  H.T.: 45/3  H.P.: 30/2 | Horas de Trabajo Independiente en el período académico y durante la **semana:** 90 |
| **Modalidad en la que se presenta el proceso de aprendizaje:**   1. **Presencial**  * Herramienta de apoyo a la presencialidad (plataforma)  1. **Distancia**  * Semipresencial (e-Learning) * Semipresencial con mediación virtual (b-Learning) * Virtual | | | |
| **Descripción del espacio pedagógico (Naturaleza y propósito):**  Este espacio formativo proporciona a los estudiantes los fundamentos de los lenguajes de programación, sus aplicaciones, semántica, e implementación. Además se espera que los estudiantes logren reconocer la definición de varios lenguajes a través de su intérprete, estudiando paso a paso distintos mecanismos, como funciones, recursión, estado, y finalmente objetos.  El objetivo principal de este espacio es el conocer los fundamentos de los lenguajes de programación, pero además generar experiencias donde de una forma colaborativa, se permita conocer y aplicar los distintos paradigmas existentes (con uno o más lenguajes de programación) para resolver problemas de la vida cotidiana. En lo general, los estudiantes podrán ver los problemas no solo desde un punto de vista sino de múltiples escenarios. | | | |
| **Capacidades previas (conocimientos, habilidades, destrezas, valores adquiridos por los estudiantes):**   1. Considera las acciones propias y de los demás para garantizar que la aplicación se integre correctamente en un entorno complejo y cumpla con las necesidades del usuario / cliente. 2. Conoce los fundamentos informáticos de la programación. 3. Analiza, diseña, y construye algoritmos para dar solución a problemas cotidianos que se le presenten 4. Plantea distintas soluciones a un problema. 5. Conoce los tipos de datos más utilizados en programación y las operaciones fundamentales sobre los mismos. 6. Aplica los mecanismos de abstracción y su importancia para solucionar problemas. 7. Maneja los fundamentos de la estructura de datos 8. Diseña y aplica de forma eficiente los diferentes tipos y estructuras de datos más adecuados para dar resolución a un 9. problema. 10. Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma seguras y eficientes, eligiendo el 11. paradigma de programación adecuado. 12. Diseña, desarrolla y evalúa aplicaciones y sistemas informáticos de forma eficiente. 13. Propone, analiza y soluciona adecuadamente problemas informáticos. | | | |
| **Competencias genéricas:**   1. Capacidad de Análisis y síntesis 2. Capacidad para plantear y resolver Problemas 3. Capacidad de comunicarse de forma oral y escrita en su lengua materna 4. Capacidad de conocer una lengua extranjera 5. Capacidad de trabajar en equipo 6. Capacidad de demostrar compromiso ético 7. Capacidad de promover el desarrollo del aprendizaje autónomo, crítico y creativo a lo largo de toda la vida. | | | |
| **Competencias específicas:**   1. Desarrollar conocimiento profundo de los principios fundamentales de los lenguajes de programación y reconoce su aplicación para interpretar, seleccionar, modelar y crear nuevos conceptos. 2. Identificar, en cualquier lenguaje de programación, los elementos comunes a la gran mayoría de ellos como ser: tipos de datos, estructuras de control, manejo de rutinas, entre otros; todo desde el estudio que fundamenta los mismos. 3. Emplear los paradigmas para crear soluciones asociados a diversos tipos de problema. | | | |
| **Subcompetencias:**   1. Conocer y comprender los principios y fundamentos que rigen los lenguajes de programación. 2. Categorizar los lenguajes de programación de acuerdo a los modelos conceptuales en los que se fundamentan, de manera que se tenga una visión más amplia del panorama de lenguajes a disposición del programador. 3. Comprende los elementos sintácticos y semánticos asociados a un lenguaje. 4. Aplicar la teoría subyacente al análisis léxico y sintáctico que hace un compilador de un lenguaje a la solución de problemas generales que involucren el uso de expresiones regulares y gramáticas libres de contexto. 5. Identificar diversos tipos de datos que pueden presentarse en los lenguajes de programación y las características asociadas a cada uno. 6. Conocer los fundamentos de los diferentes paradigmas para posteriormente aplicarlos a problemas específicos. | | | |
| **Áreas temáticas (unidades de aprendizaje o bloques):**  1. UNIDAD I: Fundamentos de Lenguajes de Programación  1.1. Conceptos básicos  1.2. Evolución de los lenguajes de programación  1.3. Clasificación de los Lenguajes de Programación  1.4. Elementos de diseño de los lenguajes  1.5. Sintaxis  1.5.1. Estructura  1.5.2. Expresiones regulares y gramáticas libres de contexto  1.5.3. Análisis sintáctico  1.6. Semántica  1.6.1. Atributos y ligaduras  1.6.2. Declaraciones y alcance  1.7. Compiladores e Intérpretes.  1.7.1. El proceso de interpretación  1.7.2. El proceso de compilación  2. UNIDAD II: Elementos comunes de los LP  2.1. Tipos de datos  2.1.1. Datos, variables y constantes  2.1.2. Tipos elementales de datos  2.1.3. Tipos estructurados de datos  2.1.4. Conversiones  2.2. Estructuras de control  2.2.1. Decisionales  2.2.2. Bucles  2.3. Subrutinas  3. UNIDAD III: Paradigmas de Programación  3.1. Programación Imperativa  3.1.1. Tipos de Datos  3.1.2. Programación modular  3.1.3. Sobrecarga  3.1.4. Recursividad  3.1.5. Lenguajes del paradigma  3.2. Programación Orientado a Objetos  3.2.1. Clases y objetos  3.2.2. Abstracción  3.2.3. Herencia  3.2.4. Encapsulamiento  3.2.5. Polimorfismo  3.2.6. Lenguajes del paradigma  3.3. Programación Lógica  3.3.1. Lógica de Predicados.  3.3.2. Metas, Clausulas, Reglas.  3.3.3. Estrategia de Búsqueda.  3.3.4. Conjunciones y Disyunciones.  3.3.5. Unificación, Recursión y Listas  3.3.6. Lenguajes del paradigma  3.4. Programación Funcional  3.4.1. Definición de funciones  3.4.2. Sistema de Tipos  3.4.3. Evaluación de expresiones  3.4.4. Recursividad  3.4.5. Lenguajes del paradigma | | | |
| **Estrategias metodológicas de aprendizaje-enseñanza:**  Este espacio será desarrollado de manera participativa y reflexiva de acuerdo a los resultados del aprendizaje esperados, haciendo uso entre otras de las estrategias metodológicas siguiente: clases magistrales, laboratorios y proyectos, trabajos individuales y en grupo, presentaciones en individuales y en grupo, uso de software afín a la temática, informes. Como complemente a la formación se realizan uso de recursos textuales (instructivos, guías, u otros), audio y video; además de las herramientas que las Tecnologías de la Información y Comunicación (TIC) permitan. | | | |
| **Resultados de aprendizaje:**   1. Domina los principios fundamentales de los lenguajes de programación y los aplica en la evaluación de un lenguaje de programación. 2. Comprende y aplica los elementos comunes a la gran mayoría de los lenguajes de programación, como ser: tipos de datos, estructuras de control, manejo de rutinas, entre otros; todo esto para aprovecharlo mejor como herramienta y reducir el tiempo de desarrollo. 3. Selecciona el mejor paradigma para dar solución a diversos tipos de problema | | | |
| **Estrategias de evaluación de los aprendizajes (diagnóstica, formativa, sumativa):**  Se incorporan los conceptos de evaluación formativa, continua, criterial, sumativa y auténtica, tanto para fines diagnósticos como formativos y para promoción. Estas formas de evaluación deben orientarse al logro de los resultados de aprendizaje. Se recomienda incorporar el uso de tecnología pertinente para favorecer no solo la recolección de evidencias de aprendizaje sino también el reporte y uso de los resultados.  a) Evaluación Diagnóstica  Se realiza al inicio del curso o al inicio de cada etapa formativa, según se requiera. Debe permite identificar el grado de logro de los resultados de aprendizajes que son requisito para los nuevos aprendizajes. Se podrá utilizar pruebas objetivas, cuestionarios y otros instrumentos o actividades que se estimen oportunos. Los resultados de esta evaluación deben ser considerados para la definición de estrategias para favorecer el logro de resultados de aprendizaje, en las cuales el estudiante asume la mayor responsabilidad con el acompañamiento docente; sin afectar de manera significativa el desarrollo del programa de este espacio pedagógico.  b) Evaluación Formativa  Se incorporan los conceptos de evaluación de proceso, criterial, continua y auténtica. Se utilizarán instrumentos coherentes con esos conceptos de evaluación: autoevaluación, coevaluación, carpeta del estudiante, talleres, laboratorios, mapas mentales, V de Gowin, entre otros; en forma individual y/o colaborativa. Los resultados de evaluación deben ser utilizados para que el estudiante, con la guía del docente, valore el logro de aprendizajes y defina acciones para fortalecerlos.  c) Evaluación Sumativa  Esta evaluación se enfoca en determinar el grado de logró de los resultados de aprendizaje de este espacio pedagógico y si el estudiante cumple el criterio de aprobación. Los método e instrumentos a utilizar deben ser coherentes con este propósito. Se sugiere el uso de pruebas objetivas, exposiciones, talleres, guías, laboratorios, proyectos, entre otros. | | | |
| **Referencias bibliográficas sugeridas:**   1. Básicas   Pratt, T. W., & Escalona García, H. J. (1998). Lenguajes de programación: Diseño e implementación.Scott, M. L. (2009).  Programming language pragmatics. (9 ed.). Burlington MA: Morgan Kaufmann.  Appleby, D., & Vandekopple, J. J. (1997). Lenguajes de programación paradigma y práctica. (2 ed.), McGraw-Hill.  Gabbrielli, M., & Martini, S. (2010). Programming languages (1st ed.). London: Springer.  Hopcroft, J., Motwani, R., & Ullman, J. (2001). Introduction to automata theory, languages, and computation (1st ed.). Boston: Addison-Wesley.   1. Complementarias   Álvarez, B., & Cáceres, D. (2008). Guía de Referencia Básica a Ada 2005 (1ra ed.). Universidad Politécnica de Cartagena. Extraído de: repositorio.bib.upct.es/dspace/handle/10317/1467  Ada Reference Manual, ISO/IEC 8652:2007(E). (2006) (3ra ed.).  Extraído de: www.adaic.org/resources/add\_content/standards/05aarm/AA.pdf  Brna, P. (2001). Prolog programming: A first course. Material no publicado.  Daniel, D. (2013). GNU PROLOG A Native Prolog Compiler with Constraint Solving over Finite Domains (1.44 ed.). Extraído de: www.gprolog.org/manual/gprolog.pdf | | | |
| **Recursos adicionales (revistas, Web, videos, películas, otros):**  Haskell  https://www.haskell.org/  Compilador GNAT para Ada:  http://www.adacore.com/gnatpro/toolsuite/  http://www.adaic.org/  http://www.ada2012.org/  Flex  https://www.cs.princeton.edu/~appel/modern/c/software/flex/flex.html  http://web.mit.edu/gnu/doc/html/flex\_1.html  Gprolog  http://www.gprolog.org/  http://www.mgencer.com/files/PrologTutorial.html  http://flex.sourceforge.net/ | | | |