Diseño de software para cómputo científico

TÍTULO: Diseño de software para cómputo científico
AÑO: 2020
CUATRIMESTRE: segundo
Docentes: Dr. Juan B. Cabral, Dr. Martín Chalela.
N° DE CRÉDITOS:
CARGA HORARIA: 30 horas de teoría y 30 horas de práctica.
CARRERA/S: Doctorado en Matemática, Doctorado en Astronomía, Doctorado en Física.

FUNDAMENTOS

Actualmente la ciencia tiene una fuerte dependencia de grandes infraestructuras computacionales tales como super-computadoras e infraestructuras de redes, por lo que es fundamental para las tareas del científico moderno desarrollar herramientas confiables, optimizando el uso del cómputo, así como su tiempo en tareas de desarrollo. En este curso se brindarán herramientas prácticas para el correcto uso de lenguajes de alto y bajo nivel, que combinan la simpleza de los primeros y la eficiencia de los segundos, y ayudan a disminuir los tiempos de desarrollo de proyectos científicos. Además, se introducirán técnicas y tecnologías modernas para la creación de aplicativos y librerías robustas confiables. Si bien la ingeniería de software es un área imposible de barrer extensivamente en su totalidad en una materia, se propone preparar al alumno en el uso eficiente de herramientas de alto nivel así como prácticas básicas para la mejora de la calidad de sus proyectos resultantes.

OBJETIVOS

- Que el alumno comprenda la necesidad de la correcta metodología de desarrollo de software para lograr herramientas computacionales confiables.
- Comprender la importancia y limitaciones de lenguajes de alto nivel para la optimización de tiempos de desarrollo.
- Brindar herramientas teórico-metodológicas para la creación y evaluación de herramientas de cómputo orientadas al àmbito científico.

PROGRAMA

Unidad 1: Lenguajes de alto nivel

Diferencias entre alto y bajo nivel.
Lenguajes dinámicos y estáticos.
Introducción al lenguaje Python.
Librerías de cómputo científico.
Orientación a objetos, decoradores.

Unidad 2: Calidad de software.

Depuración de código.
Principios de diseño: DRY y KISS
Pruebas unitarias y funcionales con pytest.

Testing basados en propiedades (Hypothesis).
Cobertura de código (codecov).
Refactoreo.
Perfilado de código (profiling) a nivel de aplicación, funciones, líneas y estadístico.
Perfiles de memoria.

Unidad 3: Persistencia de datos.

Persistencia de binarios en Python (pickle).
Archivos INI/CFG, CSV, JSON, XML y YAML.
Lectura y escritura de archivos.
Archivos de configuración.
Formato HDF5.
Bases de datos relacionales y SQL.
Breve repaso de bases de datos No relacionales.

Unidad 4: Optimización, paralelismo, concurrencia y cómputo distribuido en alto nivel.

Optimización y Optimización prematura. Cuellos de botella, Legibilidad vs. Optimización.
Compiladores justo a tiempo (numba).
Multithreading.
Paralelismo.
Concurrencia.
Cómputo distribuido con Dask.

Unidad 5: Integración con lenguajes de alto nivel con bajo nivel.

Cython.
Integración de Python con C.
Integración de Python con FORTRAN.

Unidad 6: Utilidades y distribución de paquetes.

Gestión de proyectos, y estimaciones.
Licenciamiento.
Entornos virtuales.
Interfaces de línea de comando.
Versionado.
Integración con herramientas del sistema operativo (subprocess, sh).
Integración Continua.
Paquetes python (Setuptools, Pip, PyPI, Python-Wheels y Conda).

PRÁCTICAS

Se asignará a los alumnos una herramienta a analizar y diseñar modificaciones a lo largo de toda la materia, sobre la cual se realizarán prácticos por cada unidad.

BIBLIOGRAFÍA

- Van Rossum, G. (2007, June). Python Programming Language. In USENIX annual technical conference (Vol. 41, p. 36).
- Pilgrim, M., & Willison, S. (2009). Dive Into Python 3 (Vol. 2). Apress. Astels, D. (2003). Test driven development: A practical guide. Prentice Hall Professional Technical Reference.
- Oliphant, T. E. (2007). Python for scientific computing. Computing in Science & Engineering, 9(3), 10-20.
- Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In Proceedings of the 14th Python in Science Conference (No. 130-136).
- Vasilescu, B., Van Schuylenburg, S., Wulms, J., Serebrenik, A., & van den Brand, M. G. (2014, September). Continuous integration in a social-coding world: Empirical evidence from GitHub. In 2014 IEEE International Conference on Software Maintenance and Evolution (pp. 401-405). IEEE.
- Folk, M., Cheng, A., & Yates, K. (1999, November). HDF5: A file format and I/O library for high performance computing applications. In Proceedings of supercomputing (Vol. 99, pp. 5-33).
- Lam, S. K., Pitrou, A., & Seibert, S. (2015, November). Numba: A llvm-based python jit compiler. In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC (p. 7). ACM.
- Tikir, M. M., & Hollingsworth, J. K. (2002, July). Efficient instrumentation for code coverage testing. In ACM SIGSOFT Software Engineering Notes (Vol. 27, No. 4, pp. 86-96). ACM.
- Fink, G., & Bishop, M. (1997). Property-based testing: a new approach to testing for assurance. ACM SIGSOFT Software Engineering Notes, 22(4), 74-80.

MODALIDAD DE EVALUACIÓN

La modalidad de evaluación es la de una presentación oral al finalizar el curso con la presentación de un trabajo integrador relacionado con el tema de investigación.

REQUERIMIENTOS PARA EL CURSADO

Se requiere que el alumno tenga conocimientos básicos de programación, de preferencia del lenguaje Python.