VNiVERSiDAD D SALAMANCA

ESCUELA POLITÉCNICA SUPERIOR DE ÁVILA

Máster en Geotecnologías Cartográficas en Ingeniería y Arquitectura

DESARROLLO DE HERRAMIENTAS EN PYTHON PARA CÁLCULOS GEODÉSICOS Y TOPOGRÁFICOS

Proyecto Final de Máster

Alumno: D. Juan Pedro García Tendero
Ingeniero Técnico en Topografía

Tutor E.P.S.A.: D. José Antonio Martín Jiménez

Co - tutor: D. David Hernández López
Dr. Ingeniero en Geodesia y Cartografía

Ávila, septiembre de 2011

A mis dos mujeres, mi hija Emma y mi esposa Sonia.

Agradecimientos

He de agradecer su dedicación a mis dos tutores en este Proyecto Final de Máster.

Deseo mostrar mi gratitud a D. José Antonio Martín Jiménez quien supervisó y tutorizó este Proyecto desde la Universidad de Salamanca. Escuela Politécnica Superior de Ávila.

Muy especialmente, quiero agradecer a D. David Hernández López su dedicación y apoyo en la ejecución de esta tarea. Por las tardes de verano, dedicadas a la resolución de dudas o planteamientos al proyecto y por la confianza que depositó en mí al aceptar ser mi tutor. De la misma manera, espero haber estado a la altura esperada en la consecución de los objetivos que desde el primer momento me planteó.

Resumen

Como Ingeniero Técnico en Topografía, numerosas han sido las necesidades de disponer de herramientas informáticas para los distintos cálculos necesarios desde la etapa de estudiante en la Universidad como una vez incurso en la vida laboral. Desde el simple cálculo de un acimut y distancia entre dos puntos, medidas de superficies encerradas en polígonos irregulares, datos de replanteo de puntos, cálculo de previsión de errores, o las diferentes transformaciones de coordenadas de grandes cantidades de puntos entre unas proyecciones y otras y entre distintos Sistemas de Referencia Coordenados.

Partiendo de esta necesidad y de la existencia de la librería "Geotop" en Octave del Profesor y Co-tutor de este proyecto Dr. D. David Hernández López, se han elaborado una serie de herramientas en Python para cálculos geodésicos y topográficos en Python, principalmente, migrando del lenguaje Octave a lenguaje Python.

Además del uso para cálculos topográficos en ámbitos del uso diario de ingenieros Técnicos en Topografía o cualquier Ingeniería afín, esta librería también se concibe como una herramienta de uso para el estudiante de estas disciplinas en las Universidades. Al tratarse de software libre es perfectamente editable y adaptable a las necesidades de cálculo y presentación que se consideren oportunas.

La elección de Python, aunque posteriormente se desarrolle más, se debe a que se trata de un lenguaje de programación fácil de aprender, potente y en auge. Tiene eficaces estructuras de datos de alto nivel y una solución de programación orientada a objetos eficaz. La elegante sintaxis de Python, su gestión de tipos dinámica y su naturaleza interpretada hacen de él el lenguaje ideal para guiones (scripts) y desarrollo rápido de aplicaciones, en muchas áreas y en la mayoría de plataformas.

El intérprete de Python y la extensa biblioteca estándar están disponibles libremente, en forma de fuentes o ejecutables y se pueden distribuir libremente.

TABLA DE CONTENIDOS

1 OBJETIVOS DEL TRABAJO

Dentro del marco general de la guía docente del Máster en Geotecnológias Cartográficas en Ingeniería y Arquitectura impartido por la Escuela Politécnica Superior de Ávila de la Universidad de Salamanca, se establece como requisito para la obtención de esta titulación la realización de un proyecto final de máster. A tal fin responde el presente trabajo, en el que se ha optado por la realización de un proyecto de desarrollo de herramientas en Python para cálculos geodésicos y topográficos.

Como Ingeniero Técnico en Topografía, numerosas han sido las necesidades de disponer de herramientas informáticas para los distintos cálculos necesarios desde la etapa de estudiante en la Universidad como una vez incurso en la vida laboral. Desde el simple cálculo de un acimut y distancia entre dos puntos, medidas de superficies encerradas en polígonos irregulares, datos de replanteo de puntos, cálculo de previsión de errores, o las diferentes transformaciones de coordenadas de grandes cantidades de puntos entre unas proyecciones y otras y entre distintos Sistemas de Referencia Coordenados.

Partiendo de esta necesidad y de la existencia de la librería "Geotop" en Octave del Profesor y Co-tutor de este proyecto Dr. D. David Hernández López, se han elaborado una serie de herramientas en Python para cálculos geodésicos y topográficos en Python, principalmente migrando del lenguaje Octave a lenguaje Python.

Además del uso para cálculos topográficos en ámbitos del uso diario de ingenieros Técnicos en Topografía o cualquier Ingeniería afín, esta librería también se concibe como una herramienta de uso para el estudiante de estas disciplinas en las Universidades. Al tratarse de software libre es perfectamente editable y adaptable a las necesidades de cálculo y presentación que se consideren oportunas.

La elección de Python, aunque posteriormente se desarrolle más, se debe a que se trata de un lenguaje de programación fácil de aprender y potente. Tiene eficaces estructuras de datos de alto nivel y una solución de programación orientada a objetos eficaz. La elegante sintaxis de Python, su gestión de tipos dinámica y su naturaleza interpretada hacen de él el lenguaje ideal para guiones (scripts) y desarrollo rápido de aplicaciones, en muchas áreas y en la mayoría de plataformas.

El intérprete de Python y la extensa biblioteca estándar están disponibles libremente, en forma de fuentes o ejecutables y se pueden distribuir libremente.

2 CONTEXTO DEL PROYECTO.

Dentro de los objetivos de este proyecto se encuentra el aprendizaje y desarrollo de herramientas con software libre y también la elección de Python, al tratarse de un lenguaje de programación que aunque parece novedoso, esta encontrando gran aceptación en entornos docentes y también en el desarrollo de programación en la empresa privada.

2.1 ¿POR QUÉ SOFTWARE LIBRE?

Muchísimos gobiernos de todo el mundo están empezando a ver al software libre como una poderosa herramienta para disminuir sus costos de administración, reducir la dependencia tecnológica y fomentar sus industrias locales de software.

Las razones principales que impulsan la migración de los sistemas de información de las administraciones públicas al software libre son:

- Ahorro económico. El bajo o nulo coste de los productos libres permiten a las PYMES, servicios y ampliar sus infraestructuras sin que se vean mermados sus intentos de crecimiento por no poder hacer frente al pago de cantidades en licencias
- Independencia tecnológica, mayor transparencia, aumento de la interoperabilidad, permite auditar el código fuente, etc. El secretismo tenológico es uno de los grandes frenos y desequilibrios existentes para el desarrollo en el modelo de propiedad intelectual
- Fomenta el desarrollo local y la industria nacional de software. La práctica totalidad de los concursos para desarrollo de software para la administración pública pasan por compatibilizar con productos de la factoría de Microsoft, por lo que garantiza la perpetuación e ingresos hacia Microsoft y no favorece a las empresas locales que pudieran ofrecer productos equivalentes. Además de la reducción de costes por uso de software libre, ¿qué podrían aportar esas inversiones si los beneficiados fuesen empresas del propio estado, en lugar de mandar a una compañía extranjera esas enormes cantidades de dinero?
- Facilita la adaptación a las necesidades concretas de las administraciones, en materia lingüística, legislativa, de accesibilidad e imagen.
- El conocimiento generado es público.
- Fomento de la libre competencia al basarse en servicios y no licencias. Uno de los modelos de negocio que genera el software libre es la contratación de servicios de atención al cliente. Este sistema permite que las compañías que den el servicio compitan en igualdad de condiciones a

no poseer la propiedad del producto del cual dan el servicio.

- Tratamiento seguro de los datos y la información.
- Formatos estándar. Los formatos estándar permiten una interoperatividad más alta entre sistemas, evitando incompatibilidades. Los estándares de facto son válidos en ocasiones para lograr una alta interoperatividad si se omite el hecho que estos exigen el pago de royalities a terceros y por razones de mercado no interesa que se perpetúen mucho tiempo.
- Sistemas sin puertas traseras y más seguros. El acceso al código fuente permite que tanto hackers como empresas de seguridad de todo el mundo puedan auditar los programas, por lo que la existencia de puertas traseras es ilógica ya que se pondría en evidencia y contraviene el interés de la comunidad que es la que lo genera.
- Corrección más rápida y eficiente de fallos. El funcionamiento e interés conjunto de la comunidad ha demostrado solucionar más rápidamente los fallos de seguridad en el software libre, algo que desgraciadamente en el software propietario es más difícil y costoso.
- Métodos simples y unificados de gestión de software. Actualmente la mayoría de distribuciones de linux incorporan alguno de los sistemas que unifican el método de instalación de programas, librerías, etc. Por parte de los usuarios. Así se permite el acceso a las miles de aplicaciones existentes de forma segura y gratuita a la par que evitan tener que recurrir a páginas web de dudosa ética desde las que los usuarios instalan sin saberlo spyware o virus informáticos en sus sistemas. Este sistema de acceso y gestión del software se hace prácticamente utópico si se extrapola al mercado propietario.
- Sistema en expansión. El software libre ya no es una promesa, es una realidad y se utiliza en sitemas de producción por algunas de las empresas tecnológicas más importantes como IBM, Sun Microsystems, Google, Hewlett-Packard, etc. Paradójicamente, incluso Microsoft, que posee sus propias herramientas, emplea GNU Linux en muchos de sus servidores.

Entre los países más adeptos al software libre, sus desarrollos y alcance de sus proyectos de migración se encuentran de la siguiente manera:

ALEMANIA. Alemania es el país con mayor uso del software libre del mundo. No sólo está presente en todas sus dependencias gubernamentales y universidades, sino que tiene programas multimillonarios para el desarrollo de aplicaciones libres. En 2009, destinó más de 500 millones de Euros al proyecto "Open Source and Green IT". El software libre es tan importante para los alemanes que incluso el 59% de sus empresas lo utiliza, el argumento más utilizado no es la reducción de costos, sino la posibilidad de poder modificar el código fuente.

BRASIL. Fue el primer país en migrar masivamente sus sistemas de información a alternativas open source. Se llevó a cabo en primera instancia en el estado de Río Grande, luego se realizó a escala nacional.

ARGENTINA. Existen varias iniciativas estatales que apuntan a fomentar el desarrollo de software libre. Actualmente, el software libre está implementado en algunas provincias, siendo Santa Fe el máximo exponente de utilización, dónde todo el software ha sido liberado bajo licencia GPL.

FRANCIA. La Asamblea Nacional utiliza software libre en todos sus sistemas y terminales de trabajo. Francia viene tomando impulso en el desarrollo de software open source, existen iniciativas para que el 20% del software contratado por la Administración Pública sea libre para el 2012, también ofrece beneficios fiscales para las agrupaciones de usuarios y desarrolladores de SL.

ESPAÑA. España se ha apostado como uno de los mayores impulsores mundiales del uso de software libre. Actualmente existen 200 empresas proveedoras de soluciones, 100 comunidades de usuarios y 180 centros educativos donde el software de código abierto es una realidad cotidiana. Muchos ayuntamientos y universidades ya se pasaron a sistemas open source.

A principios de 2010 se presentó un proyecto liderado por la Plataforma Tecnológica Española de Software y Servicios con el apoyo del Plan Avanza. Dicho plan, pretende colocar a España a la par del resto de Europa en el diseño y uso de software libre de confianza y calidad similares a los comerciales. Este proyecto, llamado Vulcano, trata de unificar el trabajo de distintas universidades, centros tecnológicos y empresas que, hasta el momento, habían dedicado sus esfuerzos al ámbito de la educación.

En un entrevista ofrecida por David Sánchez, Director de Comunicación y Relaciones Instituciones en CENATIC, que es el Centro Nacional de Referencia de Aplicación de las Tecnologías de la información y la Comunicación basadas en Fuentes Abiertas, podemos encontrar: http://www.muylinux.com/2011/04/07/entrevistamos-a-david-sanchez-de-cenatic/

CENATIC es una Fundación Pública Estatal del gobierno de España promovida por el Ministerio de Industria, Turismo y Comercio y la Junta de Extremadura, que además cuenta en su patronato con las comunidades autónomas de Andalucía, Aragón, Asturias, Cantabria, Cataluña, Islas Balares, País Vasco y Galicia, así como las empresas Atos Origin, Telefónica y Grupo Ápex.

Su trabajo es promover el conocimiento y uso del software libre en todos los ámbitos de la sociedad, con especial atención en las administraciones públicas, las empresas, el sector tecnológico proveedor o usuario de tecnologías libres y las comunidades de desarrollo, a través de proyectos específicos para cada uno de estos sectores.

" La apuesta a del Administración del Estado por el software libre se inició ya en los 90, cuando el entonces ministerio para las Administraciones Públicas llevó a cabo la primera gran migración a software libre de 1.375 servidores. Llegaron también proyectos como el del Principado de Asturias para generar de forma colaborativa un

marco basado en software libre, que permitió crear clúster de empresas locales capaces de aportar servicios de *eAdministracion* que de otra manera tendrían que haber contratado fuera de su comunidad."

" O el proyecto de gvSIG de la Consellería d'Infrestructures i Transport de la Generalitat Valenciana, un sistema de información geográfico que cuenta con un tejido empresarial local alrededor del proyecto y están haciendo de la Comunidad Valenciana el núcleo más activo a nivel mundial de desarrollo GIS en entornos libres. También es muy conocido el proyecto de Red Tecnológica Educativa de la Junta de Extremadura, con Linex como primera distribución regional y concepto pionero, adoptado ya por 7 comunidades autónomas."

"Finalmente, a nivel de administración local, hasta el 80% de los grandes ayuntamientos cuentan hoy en día con proyectos de software libre, siendo quizá el más destacado el del Ayuntamiento de Zaragoza".

2.2 LENGUAJES DE PROGRAMACIÓN

Un **lenguaje de programación** es un lenguaje artificial que puede ser usado para controlar el comportamiento de una máquina, especialmente una computadora. Estos, se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas.

Debe distinguirse de "lenguaje informático", que es una definición más amplia, puesto que estos incluyen otros lenguajes como son el HTML o PDF que dan formato a un texto y no es programación en sí misma.

El programador es el encargado de utilizar un lenguaje de programación para crear un conjunto de instrucciones que, al final, constituirá un programa o subprograma informático.

En su uso, un lenguaje de programación puede acercarse a la forma humana de expresarse y, por eso, este tipo de lenguajes es llamado de alto nivel. Esto significa que utilizan palabras y formas en sus estructuras que se asemejan al lenguaje natural (especialmente al inglés). En cambio, aquellos lenguajes que se aproximan más a la forma en la cual la computadora se maneja, son llamados lenguajes de bajo nivel. Esto significa que lo que el programador deba escribir se acercará al lenguaje máquina, que es, en definitiva, lo que las computadoras pueden interpretar.

De todas maneras, un lenguaje de programación difiere en muchos aspectos de un lenguaje humano. Un código escrito en un lenguaje de programación específico siempre se interpreta de la misma manera (no como los idiomas humanos ambiguos), los errores son mucho más significativos (a tal punto de que un código puede volverse ininterpretable por la computadora), etc.

El código fuente es el conjunto de instrucciones que conforman un programa (o subprograma o módulo). El código fuente debe ser compilado para poder ser interpretado y ejecutado por la computadora. La compilación traduce el código fuente (que depende del lenguaje de programación) a un lenguaje máquina (que depende del sistema de la máquina).

Existen lenguajes del tipo script que son directamente ejecutados por un intérprete y no necesitan compilación.

Los lenguajes de programación pueden clasificarse empleando distintos métodos y puntos de vista. Esta clasificación se basa en el paradigma que utilizan. Se debe aclarar que existen muchos más paradigmas y subparadigmas de programación no incluidos dentro de los mencionados. Además, todavía hay conflictos en las definiciones y alcances de ciertos paradigmas.

2.3 PARADIGMAS DE PROGRAMACIÓN

Un paradigma de programación provee (y determina) la visión y métodos de un programador en la construcción de un programa o subprograma. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas (con la solución de múltiples "problemas" se construye una aplicación).

Los lenguajes de programación son basados en uno o más paradigmas. (Véase Programación en múltiples paradigmas) Por ejemplo: Smalltalk y Java son lenguajes basados en el paradigma orientado a objetos. El lenguaje de programación Scheme, en cambio, soporta sólo programación funcional. En cambio Python, soporta múltiples paradigmas.

Clasificación por paradigmas de programación

Paradigma Imperativo: describe la programación como una secuencia instrucciones o comandos que cambian el estado de un programa. El código máquina en general está basado en el paradigma imperativo. Su contrario es el paradigma declarativo. En este paradigma se incluye el paradigma procedimental (procedural) entre otros.

Paradigma Declarativo: No se basa en el cómo se hace algo (cómo se logra un objetivo paso a paso), sino que describe (declara) cómo es algo. En otras palabras, se enfoca en describir las propiedades de la solución buscada, dejando indeterminado el algoritmo (conjunto de instrucciones) usado para encontrar esa solución. Es más complicado de implementar que el paradigma imperativo, tiene desventajas en la eficiencia, pero ventajas en la solución de determinados problemas.

Paradigma Estructurado: la programación se divide en bloques (procedimientos y funciones) que pueden o no comunicarse entre sí. Además la programación se controla con secuencia, selección e iteración. Permite reutilizar código programado y otorga una mejor compresión de la programación. Es contrario al paradigma inestructurado, de poco uso, que no tiene ninguna estructura, es simplemente un "bloque", como por ejemplo, los archivos batch (.bat).

Paradigma Orientado a Objetos: está basado en la idea de encapsular estado y operaciones en objetos. En general, la programación se resuelve comunicando dichos objetos a través de mensajes (programación orientada a mensajes). Se puede incluir -aunque no formalmente- dentro de este paradigma, el paradigma basado en objetos, que además posee herencia y subtipos entre objetos. Ej.: Simula, Smalltalk, C++, Java, Visual Basic .NET, etc.
Su principal ventaja es la reutilización de códigos y su facilidad para pensar soluciones a determinados problemas.

Paradigma Funcional: este paradigma concibe a la computación como la evaluación de funciones matemáticas y evita declarar y cambiar datos. En otras palabras, hace hincapié en la aplicación de las funciones y composición entre ellas, más que en los cambios de estados y la ejecución secuencial de comandos (como lo hace el paradigma procedimental). Permite resolver ciertos problemas de forma elegante y los lenguajes puramente funcionales evitan los efectos secundarios comunes en otro tipo de programaciones.

Paradigma lógico: se basa en la definición de reglas lógicas para luego, a través de un motor de inferencias lógicas, responder preguntas planteadas al sistema y así resolver los problemas. Ej.: prolog.

Otros paradigmas y subparadigmas son: paradigma orientado al sujeto, paradigma heurístico, paradigma reflectante, programación basada en reglas, paradigma basado en restricciones, programación basada en prototipos, etc.

2.4 ¿POR QUÉ PYTHON?

2.4.1 Características de Python

Python es un lenguaje de programación de tipo script creado por Guido van Rossum a principios de los años 90, cuyo nombre proviene del grupo "Monty Python". El objetivo es un lenguaje con una sintaxis muy limpia y con un código legible. Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa, funcional, estructurada,

etc. Otros paradigmas están soportados mediante el uso de extensiones.

Los lenguajes de script (lenguajes interpretados de muy alto nivel, como Perl y Python) gozan de creciente importancia e implantación en el mundo del software libre. Python es un lenguaje de programación fácil de aprender y potente. Dispone de eficaces estructuras de datos de alto nivel y una solución de programación orientada a objetos simple pero eficaz. La elegante sintaxis de Python, su gestión de tipos dinámica y su naturaleza interpretada hacen de él el lenguaje ideal para guiones (scripts) y desarrollo rápido de aplicaciones en muchas áreas y en la mayoría de las plataformas.

En Python no hay terminadores de sentencia (como el punto y como de C/C++/Java) ni marcas de inicio/fin de bloque (como las llaves de esos mismos lenguajes). La indentación como forma de marcar bloques elimina errores propios de los lenguajes citados y que son frecuentes en los estudiantes y también en los programadores profesionales: sentencias condicionales sin acción por añadir un punto y coma incorrectos, bucles con una sola sentencia cuando parece que hay dos o más (por omisión de llaves con un sangrado inadecuado del programa), sentencias con semántica "alterada" por usar una coma cuando corresponde un punto y coma o por omitir un punto y coma al declarar un registro antes de una función, etc. La indentación sólo resulta molesta cuando el tamaño de un bloque de cierta profundidad excede del tamaño de la ventana del editor.

Python es un lenguaje interpretado. Los lenguajes interpretados permiten ciclos de desarrollo breves (edición y ejecución) que animan a los estudiantes a experimentar. Python dispone de un entorno de ejecución que ayuda a detectar los errores (incluyendo aquellos que sólo se manifiestan en ejecución) señalándolos con mensajes muy informativos. Python ofrece, además, un entorno interactivo con el que es posible efectuar pequeñas pruebas o diseñar incrementalmente las soluciones a los problemas. La contrapartida de que se trate de un lenguaje interpretado es, obviamente, la menor velocidad de ejecución.

Python puede considerarse pseudocódigo ejecutable. Es muy expresivo y su sintaxis sencilla interfiere poco en la implementación de algoritmos, así que resulta un buen sustituto del pseudocódigo, con la ventaja de que los algoritmos codificados en Python sí son ejecutables.

Python ofrece un rico conjunto de estructuras de datos flexibles. El tipo lista de python (un vector dinámico heterogéneo) permite introducir con naturalidad el concepto de secuencia y presentar los algoritmos básicos de manejo de secuencias. Que la indexación empiece siempre en 0 ayuda a dar el salto a C, C++ o Java. El entorno de ejecución proporciona comprobación de validez de los índices, eliminando así una de las principales fuentes de problemas de C y C++. El hecho de que las listas sean

redimensionables elimina al programador la necesidad de tomar decisiones acerca de la longitud máxima de los vectores demasiado pronto. Por otra parte, Python es un lenguaje muy ortogonal: una vez se ha aprendido a manejar listas, por ejemplo, se sabe manejar cadenas, ya que ambos tipos son secuenciales y presentan conjuntos de operadores con igual nombre y semántica. Además de listas y cadenas, Python ofrece tuplas (listas inmutables) y diccionarios (vectores asociativos).

Python ofrece una amplísima colección de módulos (bibliotecas). Hay módulos para cualquier actividad imaginable: escritura de CGI, gestión de correo electrónico, desarrollo de interfaces gráficas de usuario, análisis de documentos HTML o XML, acceso a bases de datos, trabajo con expresiones regulares, etc. No es que haya que presentar todos los módulos pero sirve para ayudar a consultar la documentación de las bibliotecas disponibles favorece la programación eficiente.

Python es orientado a objetos. A diferencia de Java, Python permite una programación puramente procedimental. La orientación a objetos, aunque perfectamente soportada, es opcional (a menos, naturalmente, que se recurra a ciertos módulos en los que se definen clases). El soporte a la programación orientada a objetos es similar al de lenguajes como Samlltalk: la resolución de los nombres de método y atributos es dinámica.

El intérprete de Python y su extensa biblioteca estándar están disponibles libremente, en forma de fuentes o ejecutables, para las plataformas más importantes, en la sede web de Python, ***http://python.org*** y se pueden distribuir libremente. La misma sede contiene también distribuciones y direcciones de muchos módulos, programas y herramientas Python de terceras partes, además de documentación adicional.

Es fácil ampliar el intérprete de Python con nuevas funciones y tipos de datos implementados en C o C++ (u otros lenguajes a los que se pueda acceder desde C). Python es también adecuado como lenguaje de extensión para aplicaciones adaptables al usuario.

Algunas de las empresas que usan Python son:

- Google.
- Yahoo
- Industrial Light & Magic
- Walt Disney
- NASA
- SGI, Inc.

3 INSTRUMENTACIÓN Y SOFTWARE EMPLEADOS. PHYTON

3.1 INTRODUCCIÓN A PYTHON

Como se comentó en el punto anterior Python:

- es orientado a objetos. (realmente es multiparadigma, aceptando distintos tipos de paradigmas). A diferencia de Java, Python permite una programación puramente procedimental. La orientación a objetos, aunque perfectamente soportada, es opcional (a menos, naturalmente, que se recurra a ciertos módulo en los que se definen clases). El soporte a la programación orientada a objetos es similar a l de lenguajes como Samlltalk: la resolución de los nombres de método y atributos es dinámica.
- Python ofrece una amplísima colección de módulos (bibliotecas). Hay módulos para cualquier actividad imaginable: escritura de CGI, gestión de correo electrónico, desarrollo de interfaces gráficas de usuario, análisis de documentos HTML o XML, acceso a bases de datos, trabajo con expresiones regulares, etc.

Entre estas bibliotecas, cabe mencionar ***Numpy***:

Numpy es un módulo de Python, escrito en C, que define los vectores numéricos, las matrices y las operaciones entre ellos. Además de las funciones sobre vectores y matrices, también incluye funciones trigonométricas del tipo: sin, cos, tan, etc. ***Numpy*** es el paquete fundamental necesario para programación científica con Python.

Contiene entre otras cosa:

- Un poderoso N-dimensional objeto vector
- Sofisticadas funciones.
- Herramientas para integración con código de C, C++ y Fortram
- Útiles de algebra lineal, Transformada de Fourier y capacidades de números aleatorios.

Además de sus usos científicos obvios, ***NumPy*** también se puede utilizar como un eficiente multi-dimensional contenedor de datos genéricos. Pueden ser definidos tipos arbitrarios de datos. Esto permite integrar a la perfección y rápidamente con una amplia variedad de bases de datos.

El objeto principal de Numpy es el vector homogéneo multidimensional. Éste es, una tabla de elementos (generalmente números), todos del mismo tipo, indexados por una tupla de enteros positivos.
Por "multidimensional", entendemos que los vectores pueden tener varias dimensiones de ejes. El número de ejes se llamarán a menudo "rango".

Por ejemplo, las coordenadas de un punto en el espacio 3D sería [1, 2, 1] que representaría un vector de rango 1 de longitud 3. Otro ejemplo sería

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

que representa un array de rango 2 (2 dimensiones). Para más información mirar Numpy Glossary.

Otras librerías importantes en Python son:

Scipy es otra librería que usa Numpy. El desarrollo de Scipy comenzó en el año 2001. Sus orígenes se remontan al paquete con extensiones numéricas para Python denominado Numeric. Posteriormente apareció Numarray, con la intención de construir un paquete más flexible y de limpiar el código, aunque resultó ser más lengo para cálculos matriciales en pocas dimensiones. En el año 2005, el principal impulsor de ***Scipy***, Travis Oliphant, reunificó ambos en un único paquete que integrse las ventajas de ambos, y se denominó ***Numpy***, considerado el núcleo de Scipy. ***Scipy*** en sí mismo se concibe actualmente como una extensión de las funcionalidades de Numpy.

Scipy posee módulos para optimización de funciones, integración, funciones especiales, resolución de ecuaciones diferenciales ordinarias y otros muchos aspectos.

Puede ser usado con Linux, Windows y ha sido también compilado para Sun y Mac.

Scipy es un proyecto de software libre que ha sido patrocinado por una compañía, Enthought inc.

Su organización se estructura en subpaquetes, que se pueden considerar especializados en dominios científicos determinados. Podemos encontrar estos paquetes, según la ayuda de scipy (v0.4.6):

- stats -- Statistical Functions
- sparse -- Sparse matrix
- lib -- Python wrappers to external libraries
- linalg -- Linear algebra routines
- signal -- Signal Processing Tools
- misc -- Various utilities that don't have another home
- interpolate -- Interpolation Tools [*]
- optimize -- Optimization Tools [*]
- cluster -- Vector Quantization / Kmeans [*]
- fftpack -- Discrete Fourier Transform algorithms [*]

- io -- Data input and output [*]
- maxentropy -- Routines for fitting maximum entropy models [*]
- integrate -- Integration routines [*]
- lib.lapack -- Wrappers to LAPACK library [*]
- special -- Special Functions [*]
- lib.blas -- Wrappers to BLAS library [*]

Matplotlib es una librería de Python que facilita la publicación de calidad de la publicación interactiva. Permite obtener gráficas de calidad para publicaciones. Su principal submódulo para dibujar es pyplot

Mayavi

Ipython

PIL (Python Imaging Library)

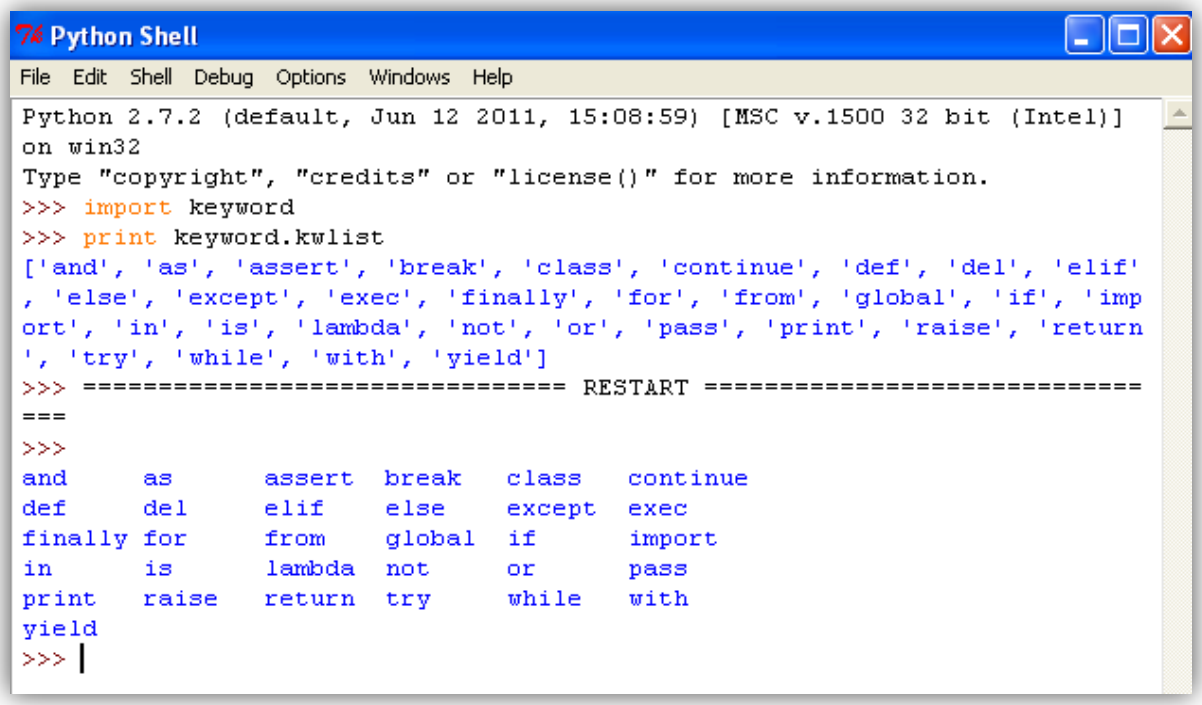
Pythonxy

EPD (Enthought python distribution $$)
…

Este texto como Memoria de un Proyecto Final de Máster no pretende ser, ni mucho menos, un manual de Python, pero sí son necesarios unos conocimientos previos, primero para comprender la filosofía del programador y posteriormente saber las utilidades que podemos obtener al trabajar con Python.

Para ello vamos a incluir algunos detalles incluidos entro de la Guía de aprendizaje de Python por Guido Van Rossum v.2.4, creador de Python. Para la realización de este documento se ha consultado como aparece en Bibliografía y Referencias en la Web numerosa documentación pero como punto de partida, parece lógico recoger la documentación de la Guía de aprendizaje del propio creador de Python.

Python contiene una serie de palabras clave, dependientes de la versión de instalación, para asegurarnos de las palabras clave de la versión descargada podemos introducir el siguiente código:



Palabras clave de Python

De la misma manera, Python trae consigo ciertas funciones que vienen de serie, esto es, no es necesario cargarlas desde ningún módulo. Dichas funciones dependen de la versión de Python instalada. En la página **http://docs.python.org/library/functions.html.** Nos encontramos con dicha relación. A modo de resumen, éstas son:

(ver página siguiente)

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	apply()
delattr()	help()	next()	setattr()	buffer()
dict()	hex()	object()	slice()	coerce()
dir()	id()	oct()	sorted()	intern()

Funciones internas de Python

3.1.1 Guía de aprendizaje de Python por Guido Van Rossum v.2.4

Extractos del documento:

Si en alguna ocasión hemos escrito un guión para un intérprete de órdenes (o shell script) de UNIX largo, puede que hayamos sentido: que nos encantaría añadir una característica más, pero ya es tan lento, tan grande, tan complicado… O la característica involucra una llamada al sistema u otra función accesible sólo desde C. El problema en sí no suele ser tan complejo como para transformar el guión en un programa en C. Igual el programa requiere cadenas de longitud variable u otros tipos de datos (como listas ordenadas de nombres de fichero) fáciles en sh, pero tediosas en C. o quizá no tengamos tanta soltura con C.
Otra situación: Quizá tengas que trabajar con bibliotecas de C diversas y el ciclo normal en C de escribir-compilar-probar-recompilar es demasiado lento. Necesitas desarrollar software con más velocidad. Posiblemente has escrito un programa al que vendría bien un lenguaje de extensión y no quieres diseñar un lenguaje, escribir y depurar el intérprete y adosarlo a la aplicación.

En tales casos, Python puede ser el lenguaje que necesitas. Python es simple, pero es un lenguaje de programación real. Ofrece más apoyo e infraestructura para programas

grandes que el intérprete de órdenes. Por otra parte, también ofrece mucho más, comprobación de errores que C y, al ser un lenguaje de muy alto nivel, tiene incluidos tipos de datos de alto nivel, como matrices flexibles y diccionarios, que llevarían días de programación en C. Dados sus tipos de datos más generales, se puede aplicar a un rango de problemas más amplio que Awk o incluso Perl, pero muchas cosas son, al menos, igual de fáciles en Python que en esos lenguajes.

Python te permite dividir su programa en módulos reutilizables desde otros programas en Python. Viene con una gran colección de módulos estándar que puedes utilizar como base de tus programas (o como ejemplos para empezar a aprender Python). También hay módulos incluidos que proporcionan E/S de ficheros, llamadas al sistema, "sockets" y hasta interfaces gráficas con el usuario, como Tk.

Python es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interacivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrrollo del prgram de la base hacia arriba. También es una calculadora muy útil.

Python permite escribir programas muy compactos y legibles. Los programas escritos en Python son normalmente mucho más cortos que sus equivalentes en C o C++, por varios motivos:

- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola sentencia.
- El agrupamiento de sentencias se realiza mediante sangrado (indentación) en lugar de begin/end o llaves.
- No es necesario declarar los argumentos ni las variables.

Python es *ampliable*: si ya sabes programar en C, es fácil añadir una nueva función o módulo al intérprete, para realizar operaciones críticas a la máxima velocidad o para enlazar programas en Python con bibliotecas que sólo están disponibles en forma binaria (como bibliotecas de gráficos específicas del fabricante). Una vez enganchado, puedes enlazar el intérprete de Python a una aplicación escrita en C y utilizarlo como lenguaje de macros para dicha aplicación.

3.1.1.1 Llamar al intérprete

En UNIX, el intérprete de Python se suele instalar como '/usr/local/bin/python' en aquellas máquinas donde esté disponible. En Windows, se instala en el directorio 'Archivos de programa' o en cualquier otro directorio seleccionado. Poner este directorio en la ruta de ejecutables hace posible arrancarlo tecleando en el intérprete de órdenes la orden:

Python

Como la elección del directorio donde reside el intérprete es una opción de instalación, es posible que se halle en otros lugares. Consulta con tu guru de Python local o tu administrador de sistemas (por ejemplo, '/usr/local/python' es una alternativa recuente).

Teclear un carácter fin de fichero (Control-D en UNIX, Control-Z en DOS o Windows) en el intérprete causa la salida del intérprete con un estado cero. Si eso no funciona, se puede salir del intérprete tecleando las siguientes órdenes: 'import sys; sys.exit()'.
Las opciones de edición de la línea de órdenes no son muy destacables. En UNIX, es posible que quien instalara el intérprete en su sistema incluyera soporte para la biblioteca de GNU 'readline', que permite una edición de línea más elaborada y la recuperación de órdenes anteriores. El modo más rápido de ver si hay soporte de edición de líneas es teclear Control-P en cuanto aparece el intérprete. Si pita, la edición de líneas está disponible. Si no sale nada o sale ^P, no está disponible la edición de líneas y sólo se puede utilizar la tecla de borrado para borrar el último carácter tecleado.

El intérprete funciona como el intérprete de órdenes de UNIX: cuando se lo llama con la entrada estándar conectada a un dispositivo tty, lee y ejecuta las órdenes interactivamente; cuando se le da un nombre de fichero como argumento o se le da un fichero como entrada estándar, lee y ejecuta un guion desde ese fichero.
Otro modo de arrancar el intérprete es '**python -c** *orden* [argumento] ...', que ejecuta las sentencias de *orden*, de forma análoga a la opción **-c** de la línea de órdenes. Como las sentencias de Python suelen contener espacios u otros caracteres que la línea de órdenes considera especiales, lo mejor es encerrar *orden* entre dobles comillas por completo.

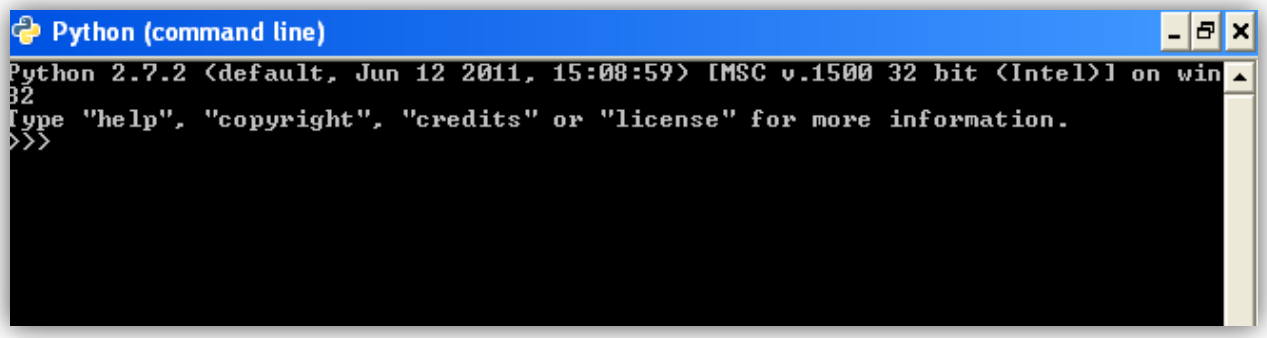


Imagen de la pantalla al abrir Python

Hay módulos de Python que son útiles como programas independientes. Se los puede llamar mediante "python-m módulo[arg] …", que ejecuta el fichero de código fuente de module como si se hubiera dado el nombre completo en la línea de órdenes.
Cuando se leen órdenes desde una tty, se dice que el intérprete está en modo interacitov. En este modo, espera a la siguiente orden con el indicador principal, que suele ser tras signos "mayor" (">>>"). Para las líneas adicionales, se utiliza el indicador secundario, por omisión tres puntos ("…").

Programar en Python puede hacerse de varias maneras según la necesidad o el gusto de cada persona. Para los neófitos mi recomendación es que utilicen el ambiente gráfico interactivo llamado ***IDLE***. Esta herramienta viene incluída con el módulo tkinter. Además de resaltar la sintaxis en colores, permite editar archivos fuente y es más amigable al inicio.

El ***IDLE*** tiene dos ambientes: el shell interactivo con título **Python Shell** en su ventana; muestra el prompt >>> y espera un comando, y uno o más editores que se abren con el menú File --> New Window. Cada editor empieza con el título *Untitled* en su ventana, el cual cambia hasta que se salva a un archivo con File --> Save As (y subsecuentemente File --> Save). Cada editor nos permite ejecutar el código Python que contiene.

Se recomienda crear una carpeta para realizar y guardar los ejemplos. Para correr *idle*, cambiar primero a esa carpeta y entonces correr idle: En MS- Windows:

```
C:\ejemplos> C:\python22\idle\idle
```

En Linux:

```
[usuario@pc ejemplos]$ idle &
```

La primera vez que hacemos un ejemplo, hemos de intentar hacerlo paso a paso en forma interactiva en el shell, tecleando cada comando. Es la forma en que aprendemos más que si simplemente copiamos y pegamos.
Una vez que tecleamos y funcionan las cosas, podemos copiar del shell interactivo y pegamos a una ventana de editor y salvamos en un archivo con terminación .py para que conservemos lo que hicimos para siempre.

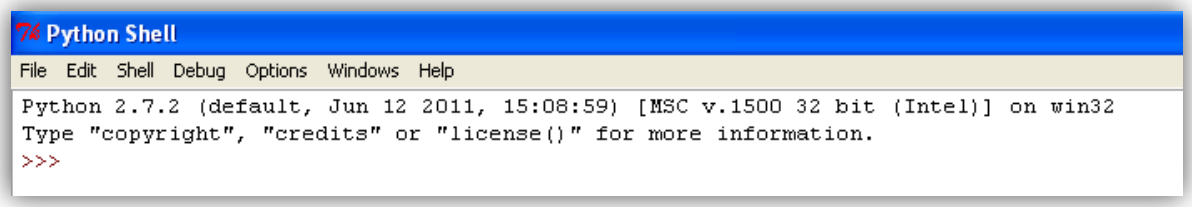
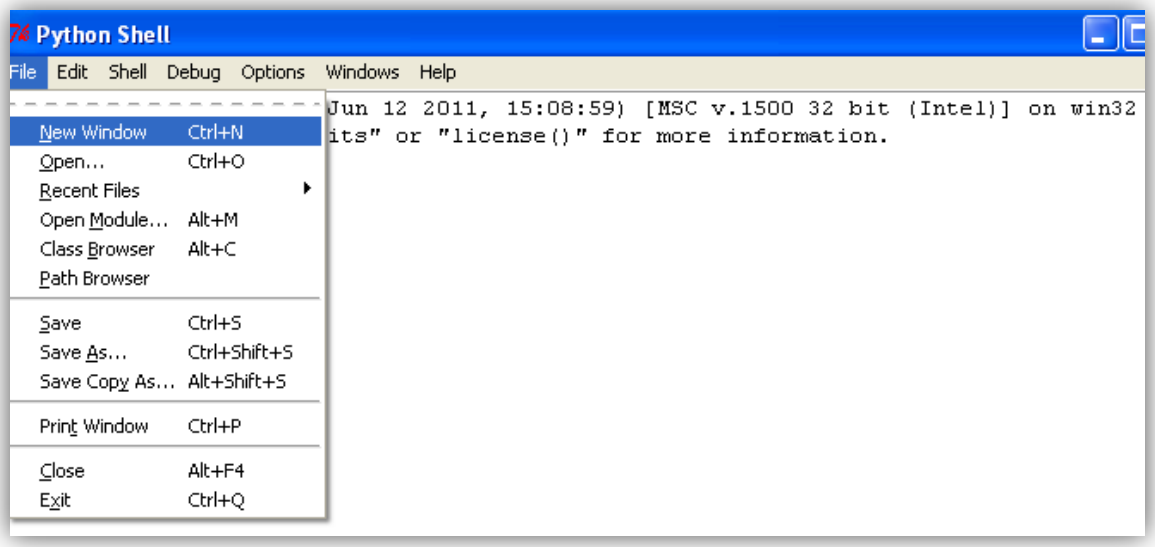
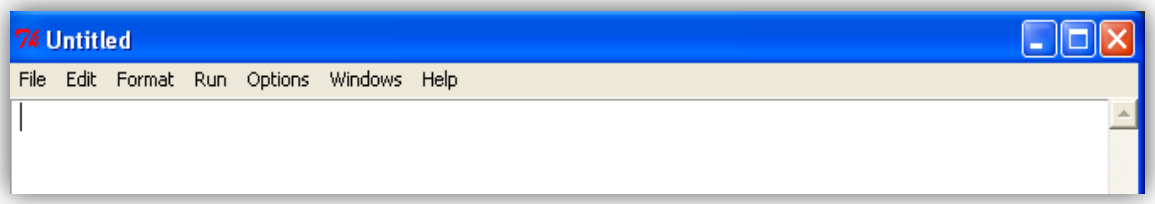


Imagen del Shell interactivo de IDLE

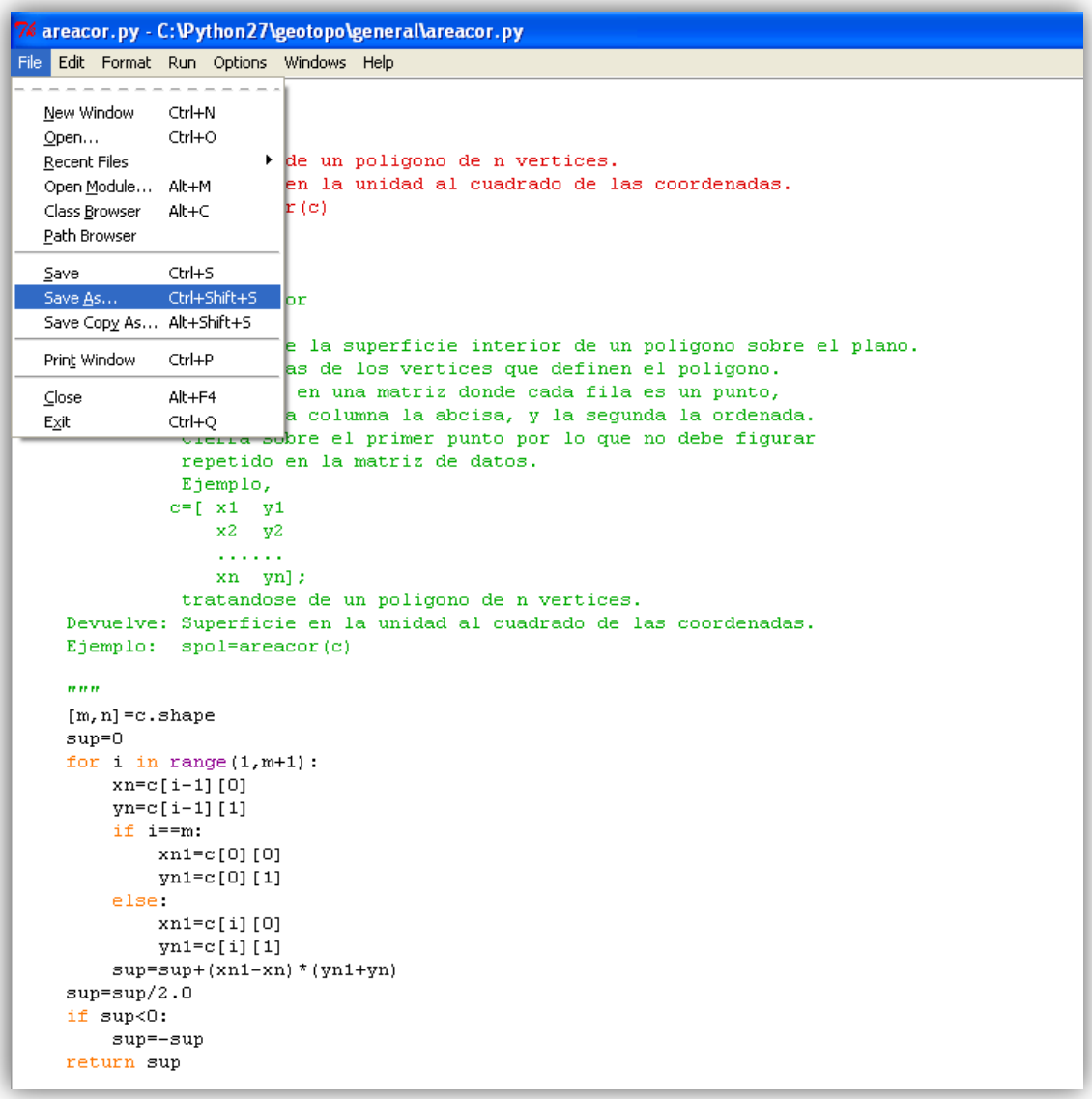


Acceso a nueva ventana dentro del Shell interactivo de IDLE



Nueva ventana del editor de IDLE

Una vez que partiendo de una ventana vacia, tenemos nuestro código, hemos de seleccionar File/Save As para guardar el archivo y tenerlo disponible



Ventana del editor de IDLE con código preparado para guardar

Una vez que nos hemos asegurado de que tenemos el código guardado ¿cómo hacemos para ejecutar el programa?. La respuesta es seleccionando Run Module. En caso de seleccionar Run Module antes de guardar el código, Python nos pide que lo guardemos antes…

Al seleccionar Run Module, Python irá interpretando línea por línea y mostrando los resultados a través del Intérprete. Si existiese algún error de sintaxis, se avisa del tipo de error y dónde se produce. Una vez corregido el problema, se vuelve a intentar el Run Module.

El tener un intérprete permite ir probando partes de programas sobre las que no se está seguro y ver cómo reaccionan, para poder decidir si se incluyen o no, y cómo se pueden adaptar. Así, se está continuamente alternando entre el intérprete y el programa. Más información en http://docs.python.org/library/idle.html.

3.1.1.2 Módulos

Si salimos del intérprete de Python y vuelves a entrar, las definiciones que se hayan hecho (funciones y variables) se pierden. Por ello, si se quiere escribir un programa algo más largo, será mejor que se utilice un editor de texto para preparar la entrada del intérprete y ejecutarlo con ese fichero como entrada. Esto se llama crear un guión. Según vayan creciendo los programas, puede que quieras dividirlos en varios ficheros para facilitar el mantenimiento. Puede que también quieras utilizar una función que has escrito en varios programas sin tener que copiar su definición en cada programa.
Para lograr esto, Python tiene un modo de poner definiciones en un fichero y utilizarlas en un guión o en una instancia interactiva del intérprete. Tal fichero se llama módulo; las definiciones de un módulo se pueden importar a otros módulos o al módulo principal (la colección de variables accesible desde un guión ejecutado desde el nivel superior y en el modo de calculadora).

Un módulo es un fichero que contiene definiciones y sentencias de Python. El nombre del fichero es el nombre del módulo con el sufijo "py". Dentro de un módulo, el nombre del módulo (como cadena) es accesible mediante la variable global __name__. Un módulo puede contener sentencias ejecutables además de definiciones de funciones. Estas sentencias sirven para inicializar el módulo. Sólo se ejecutan la *primera* vez que se importa el módulo en alguna parte1.

Cada módulo tiene su propia tabla de símbolos, que utilizan todas las funciones definidas por el módulo como tabla de símbolos global. Por ello, el autor de un módulo puede utilizar variables globales dentro del módulo sin preocuparse por conflictos con las variables globales de un usuario del módulo. Por otra parte, si sabes lo que haces, puedes tocar las variables globales de un módulo con la misma notación utilizada para referirse a sus funciones, nombreMod.nombreElem.

Los módulos pueden importar otros módulos. Es una costumbre no obligatoria colocar todas las sentencias "*import*" al principio del módulo (o guión). Los nombres del módulo importado se colocan en la tabla de símbolos global del módulo (o guión) que lo importa.
Existe una variación de la sentencia import que importa los nombres de un módulo directamente a la tabla de símbolos del módulo que lo importa. Por ejemplo:

```
>>> from fibo import fib, fib2
```

```
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Esto no introduce el nombre del módulo del que se toman los elementos importados en la tabla de símbolos local (por lo que, en el ejemplo, no está definido fibo). Además, existe una variación que importa todos los nombres que define un módulo:

```
>>> from fibo import *
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Esto importa todos los nombres, excepto los que empiezan por un guión bajo (_).

3.1.1.3 El camino de búsqueda de módulos

Cuando se importa un módulo denominado fiambre, el intérprete busca un fichero denominado 'fiambre.py' en el directorio actual y, luego, en la lista de directorios especificada por la variable de entorno PYTHONPATH. Tiene la misma sintaxis que la variable de línea de órdenes PATH de UNIX, que es una lista de nombres de directorios.

Cuando PYTHONPATH no tiene ningún valor o no se encuentra el fichero, se continúa la búsqueda en un camino dependiente de la instalación. En UNIX, normalmente es '.:/usr/local/lib/python'.

En realidad, se buscan los módulos en la lista de directorios dada por la variable sys.path, que se inicializa desde el directorio que contiene el guión de entrada (o el directorio actual), PYTHONPATH y el valor por omisión dependiente de la instalación. Esto permite que los programas que saben lo que hacen modifiquen o reemplacen el camino de búsqueda de módulos. Obsérvese que, como el directorio que contiene el guión bajo ejecución está en el camino de búsqueda de módulos, es importante que el módulo no tenga el mismo nombre que un módulo estándar, o Python lo intentará cargar el guión como módulo cuando se importe el módulo. Normalmente, esto provocará errores.

3.1.1.4 Ficheros Python "Compilados"

Como mejora considerable del tiempo de arranque de programas cortos que utilizan muchos módulos estándar, si existe un fichero llamado 'fiambre.pyc' en el directorio donde se encuentra 'fiambre.py', se supone que contiene una versión previamente "compilada a byte" del módulo fiambre. La fecha y hora de la versión de 'fiambre.py'

utilizada para generar 'fiambre.pyc' se graba en 'fiambre.pyc' y no se considera el fichero '.pyc' si no concuerdan.
Normalmente, no hay que hacer nada para generar el fichero 'fiambre.pyc'. Siempre que 'fiambre.py' se compile sin errores, se hace un intento de escribir la versión compilada a 'fiambre.pyc'. No se provoca un error si falla el intento. Si por cualquier motivo no se escribe completamente el fichero, el fichero 'fiambre.pyc' resultante será reconocido como no válido y posteriormente ignorado. El contenido del fichero 'fiambre.pyc' es independiente de la plataforma, por lo que se puede compartir un directorio de módulos entre máquinas de diferentes arquitecturas.

3.1.1.5 Módulos estandar.

Python viene con una biblioteca de módulos estándar, descrita en un documento aparte, la *Referencia de las bibliotecas*. Algunos módulos son internos al intérprete y proporcionan acceso a las operaciones que no son parte del núcleo del lenguaje pero se han incluido por eficiencia o para proporcionar acceso a primitivas del sistema operativo, como las llamadas al sistema. El conjunto de dichos módulos es una opción de configuración que también depende de la plataforma subyacente. Por ejemplo, el módulo amoeba sólo se proporciona en sistemas que de algún modo tienen acceso a primitivas Amoeba. Hay un módulo en particular que merece una especial atención, el módulo sys, que es siempre interno en cualquier intérprete de Python.

Estas variables sólo están definidas si el intérprete está en modo interactivo.
La variable sys.path es una lista de cadenas que determina el camino de búsqueda de módulos del intérprete.
Se inicializa a un valor por omisión tomado de la variable de entorno PYTHONPATH o de un valor por omisión interno, si PYTHONPATH no tiene valor. Se puede modificar mediante operaciones de lista estándar, por ejemplo:

```
>>> import sys
>>> sys.path.append('/ufs/guido/lib/python')
```

3.1.1.6 Paquetes.

Los paquetes son un método de estructurar el espacio nominal de módulos de Python, mediante el uso de "nombres de módulos con punto". Por ejemplo, el nombre de módulo A.B hace referencia a un submódulo denominado "B" de un paquete denominado "A". Del mismo modo que el uso de módulos evita que los autores de diferentes módulos tengan que preocuparse de los nombres de variables globales de los otros, la utilización de nombres de módulo con puntos evita que los autores de paquetes multi-módulo, como ***Numpy*** o ***Pil*** (Biblioteca de tratamiento de imagen de

python), tengan que preocuparse de los nombres de los módulos ajenos.

Supón que deseas diseñar una colección de módulos (un paquete) para tratar de manera uniforme ficheros de sonido y datos de sonido. Existen muchos formatos de fichero de sonido (que se suelen distinguir por la extensión, como '.wav', '.aiff' o '.au'), por lo que podrías necesitar crear y mantener una colección creciente de módulos de conversión entre los diferentes formatos. También existen muchas operaciones posibles sobre los datos de sonido (tales como mezclar, añadir eco, ecualizar o generar un efecto artificial de estereofonía), por lo que, además, estarías escribiendo una serie de módulos interminable para realizar estas operaciones. He aquí una posible estructura de tu paquete (expresado en términos de sistema de ficheros jerárquico):

```
Sonido/ Paquete de nivel superior
__init__.py Inicializa el paquete de sonido

      Formatos/ Subpaquete de conversiones de formato de ficheros
      __init__.py
      leerwav.py
      escriwav.py
      leeraiff.py
      escriaiff.py
      leerau.py
      escriau.py
      ...
      Efectos/ Subpaquete de efectos de sonido
      __init__.py
      eco.py
      surround.py
      inverso.py
      ...
      Filtros/ Subpaquete de filtros
      __init__.py
      ecualizador.py
      vocoder.py
      karaoke.py
      ...
```

Modelo de estructura paquete.módulo

Al importar el paquete, Python rastrea los directorios de sys.path buscando por el subdirectorio de paquetes.

Los ficheros '__init__.py' son necesarios para que Python trate los directorios como contenedores de paquetes. Se hace así para evitar que los directorios con nombres

comunes, como 'test', oculten accidentalmente módulos válidos que aparezcan más tarde dentro del camino de búsqueda. En el caso más sencillo, '__init__.py' puede ser un fichero vacío, pero también puede ejecutar código de inicialización del paquete o actualizar la variable __all__, descrita posteriormente.
Los usuarios del paquete pueden importar módulos individuales del paquete, por ejemplo:

```
import Sonido.Efectos.eco
```

De este modo se carga el submódulo Sonido.Efectos.eco. Hay que hacer referencia a él por su nombre completo:

```
Sonido.Efectos.eco.filtroeco(entrada, salida, retardo=0.7, aten=4)
```

Un modo alternativo de importar el submódulo es:

```
from Sonido.Efectos import eco
```

Así también se carga el submódulo eco y se hace disponible sin su prefijo de paquete, por lo que se puede utilizar del siguiente modo:

```
eco.filtroeco(entrada, salida, retardo=0.7, aten=4)
```

Y otra variación es importar la función o variable deseada directamente:

```
from Sonido.Efectos.eco import filtroeco
```

De nuevo, se carga el submódulo eco, pero se hace la función filtroeco disponible directamente:

```
filtroeco(entrada, salida, retardo=0.7, aten=4)
```

Observa que al utilizar from *paquete* import *elemento*, el elemento puede ser tanto un submódulo (o subpaquete) del paquete como cualquier otro nombre definido por el paquete, como una función, clase o variable. La sentencia import comprueba primero si el elemento está definido en el paquete. Si no, asume que es un módulo e intenta cargarlo. Si no lo consigue, se provoca una excepción ImportError.

Sin embargo, cuando se utiliza la sintaxis;

`import.elemento.subelemento.subsubelemento`, cada elemento menos el último debe ser un paquete. El último elemento puede ser un módulo o un paquete,

pero no una clase, función o variable definida en el nivel superior.

3.1.1.7 Importar * de un paquete.

Y ¿qué ocurre cuando el usuario escribe *from Sonido.Efectos import **? En teoría, debería rastrearse el sistema para encontrar qué submódulos existen en el paquete e importarlos todos. Por desgracia, esta operación no funciona muy bien en las plataformas Windows y Mac, en las que el sistema de ficheros no tiene una idea muy precisa de las mayúsculas de un fichero. En estas plataformas, no hay un modo garantizado de conocer si un fichero 'ECO.PY' debería ser importado como eco, Eco o ECO (por ejemplo, Windows 95 tiene la molesta costumbre de mostrar todos los nombres de fichero con la primera letra en mayúscula). La restricción de nombres de fichero DOS (8+3) añade otro problema para los nombres de módulo largos.
La única solución es que el autor del paquete proporcione un índice explícito del paquete. La sentencia import utiliza la siguiente convención: Si el código del '__init__.py' de un paquete define una lista llamada __all__, se considera que es la lista de nombres de módulos que se deben importar cuando se encuentre from *paquete* import *. Depende del autor del paquete mantener la lista actualizada cuando se libere una nueva versión del paquete. Los autores del paquete pueden decidir no mantenerlo, si no es útil importar * del paquete. Por ejemplo, el fichero 'Sonido/Efectos/__init__.py' podría contener el siguiente código:

```
__all__ = ["eco", "surround", "inverso"]
```

Esto significaría que from Sonido.Efectos import * importaría los tres submódulos mencionados del paquete Sonido.
Si __all__ no está definido, la sentencia *from Sonido.Efectos import * no* importa todos los módulos del subpaquete Sonido.Efectos al espacio nominal actual. Sólo se asegura de que el paquete Sonido.Efectos ha sido importado (ejecutando posiblemente el código de inicialización de '__init__.py') y luego importa cualesquiera nombres definidos en el paquete. Esto incluye cualquier nombre definido (y submódulos cargados explícitamente) por '__init__.py'. También incluye cualquier submódulo del paquete explícitamente importado por sentencias import anteriores.

Mira este código:

```
import Sonido.Efectos.eco
import Sonido.Efectos.surround
from Sonido.Efectos import *
```

En este ejemplo, los módulos eco y surround se importan al espacio nominal vigente porque están definidos en el paquete Sonido.Efectos cuando se ejecuta la sentencia

`from...import (esto también funciona si está definido __all__).`

Observa que en general se debe evitar importar * de un módulo o paquete, ya que suele dar como resultado código poco legible. Sin embargo, se puede usar para evitar teclear en exceso en sesiones interactivas y cuando ciertos módulos estén diseñados para exportar sólo nombres que cumplan ciertas reglas.

Recuerda, no hay nada incorrecto en utilizar `from Paquete import submódulo_concreto`. De hecho, es la notación recomendada salvo que el módulo que importa necesite usar submódulos del mismo nombre de diferentes paquetes.

3.1.1.8 Referencias internas al paquete.

Es común que los submódulos necesiten hacerse referencias cruzadas. Por ejemplo, el módulo surround podría utilizar el módulo eco. De hecho, tales referencias son tan comunes que la sentencia import busca antes en el paquete contenedor que en el camino de búsqueda de módulos estándar. Por ello, basta con que el módulo surround use *import eco* o *from eco import filtroeco*. Si el módulo importado no se encuentra en el paquete actual (el paquete del que el módulo actual es submódulo), la sentencia import busca un módulo de nivel superior con el nombre dado.
Cuando se estructuran los paquetes en subpaquetes (como el paquete Sonido del ejemplo), no hay un atajo para referirse a los submódulos de los paquetes hermanos y se ha de utilizar el nombre completo del subpaquete.
Por ejemplo, si el módulo Sonido.Filtros.vocoder necesita utilizar el módulo eco del paquete Sonido.Efectos, debe utilizar from Sonido.Efectos import eco.

3.2 FUNCIONES INCLUIDAS EN NUMPY POR CATEGORÍAS (ARRAY CREATION)

A modo indicativo e introductorio, se incluyen las funciones incluidas en la librería Numpy por categorías. Ver más información en http://docs.scipy.org/doc/numpy/.

`Numerical`

- `arange(), arrayrange()`
- `linspace(), logspace()`

`Ones and zeros`

- `empty(), empty_like()`
- `eye(), identity()`
- `ones(), ones_like()`

- tri(), tril(), triu()
- zeros(), zeros_like()

From existing data

- array(), recarray()
- asarray(), asanyarray(), asmatrix()
- copy()
- fromarrays(), frombuffer(), fromfile(), fromfunction(), fromiter(), loadtxt()

Building matrices

- bmat()
- diag(), diagflat()
- mat(), matrix()
- vander()

Reinterpreting arrays

- view()

Types

- astype()
- cast[]()
- int8(), int16(), int32(), int64(), int128()
- uint8(), uint16(), uint32(), uint64(), uint128()
- float16(), float32(), float64(), float96(), float128(), float256()
- complex32(), complex64(), complex128(), complex192(), complex256(), complex512()
- bool_()
- object_()
- void(), str_(), unicode_()
- byte(), ubyte()
- short(), ushort()
- intc(), uintc()
- intp(), uintp()
- int_(), uint()
- longlong(), ulonglong()
- single(), csingle()
- float_(), complex_()

- longfloat(), clongfloat()

Kinds of array

- asarray()
- asanyarray()
- asmatrix()

Changing the number of dimensions

- atleast_1d(), atleast_2d(), atleast_3d()
- broadcast()
- expand_dims()
- squeeze()

Transpose-like operations

- rollaxis()
- swapaxes()
- T, transpose()

Reshaping arrays

- flat, flatten()
- ravel()
- reshape(), shape

Array modification

Joining arrays

- append()
- column_stack()
- concatenate()
- dstack(), hstack(), vstack()

Splitting arrays

- array_split()
- dsplit(), hsplit(), split(), vsplit()

Enlarging arrays

- tile()
- repeat()

Adding and removing elements

- delete()
- insert()
- resize()
- trim_zeros()

- unique()

Rearranging elements

- fliplr(), flipud()
- reshape()
- roll()
- rot90()

Indexing

- []
- take()
- put()
- putmask()

Indexing syntax

- ...
- slice()
- newaxis
- index_exp[]

Generating arrays suitable for indexing

- c_[]
- r_[]
- s_[]
- nonzero()
- where()
- indices()
- ix_()
- mgrid[]
- ogrid()

Indexing-like operations

- choose()
- where()
- compress()
- diag(), diagonal()
- select()

Iterating

- flat

- ndenumerate()
- ndindex()

Logic

- []
- all(), any()
- allclose()
- alltrue()
- nonzero()
- piecewise()
- sometrue()

Finding things

- argmax(), argmin()
- searchsorted()

Array statistics

- average(), mean()
- bincount(), histogram()
- corrcoef()
- cov()
- max(), min(), ptp()
- median()
- std(), var()

ufuncs

- abs(), absolute()
- add(), multiply()
- angle()
- arccos(), arcsin(), arctan()
- arccosh(), arcsinh(), arctanh()
- arctan2()
- bitwise_and(), bitwise_or(), bitwise_xor()
- ceil(), floor(), round()
- conj(), conjugate()
- cos(), sin(), tan()
- cosh(), sinh(), tanh()
- fix()

- hypot()
- logical_and(), logical_not(), logical_or(), logical_xor()
- maximum(), minimum()

ufunc methods

- accumulate()
- outer()
- reduce()

Functional constructs

- apply_along_axis()
- apply_over_axis()
- vectorize()

Random numbers

- beta(), binomial(), gumbel(), poisson(), standard_normal(), uniform(), vonmises(), weibull()
- bytes()
- permutation()
- rand(), randint(), randn()
- random_integers()
- random_sample()
- ranf()
- sample()
- seed()
- shuffle()

Array math

- clip()
- cross()
- cumprod(), cumsum()
- diff()
- digitize()
- dot()
- inner()
- outer()
- inv(), pinv()
- poly1d()
- polyfit()

- prod(), sum()
- tensordot()
- vdot()

FFT

- fft()
- fftfreq()
- fftshift()
- ifft()

Linear algebra

- inv()
- lstsq()
- solve()
- svd()
- trace()

Array output

- savetxt()
- set_printoptions()
- tofile()
- tolist()

Other

- sort(), argsort()
- binary_repr()
- dtype()
- fill()
- finfo()
- generic
- imag, real
- inf, nan
- item()
- lexsort()
- ndim
- shape
- typeDict()

Numpy Functions by Category (última edición 2008-06-13 12:26:59 efectuada por jh)

- MoinMoin Powered
- Pyt

4 METODOLOGÍA

En este apartado se proporciona una descripción detallada de la metodología utilizada para el desarrollo e implementación de la biblioteca.

Utilizando un paradigma Estructurado de la programación, que considera que la programación se divide en bloques (procedimientos y funciones) que, pueden o no, comunicarse entre sí. Esto, nos permite reutilizar código programado y nos proporciona una mejor comprensión de la programación. En el último paquete de la librería, también se ha realizado una incursión a la programación orientada a objetos con la declaración de algunas clases.

4.1 ESTRUCTURA DE LA LIBRERÍA GEOTOPO

En nuestro caso hemos partido del concepto de creación de una librería de funciones del ámbito de la Topografía y Geodesia. Esta librería de Python se estructura como un paquete con el nombre de la librería "**geotopo**", que a su vez contiene 6 paquetes más, según tipología. Así, tenemos:

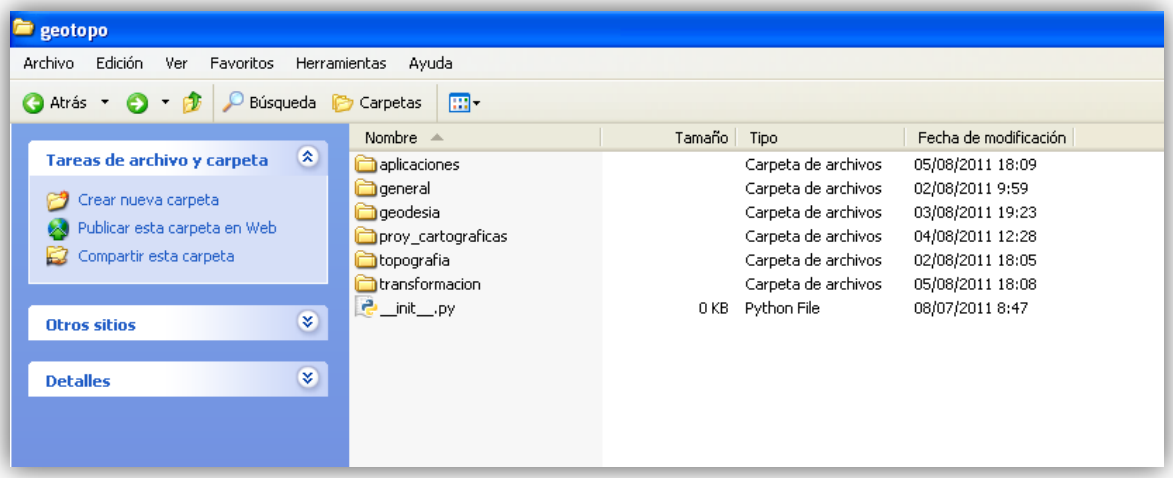
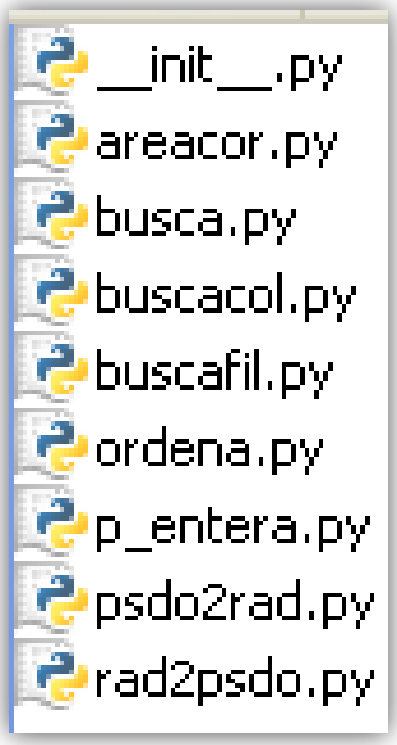


Imagen de los paquetes incluidos dentro del paquete (librería) geotopo

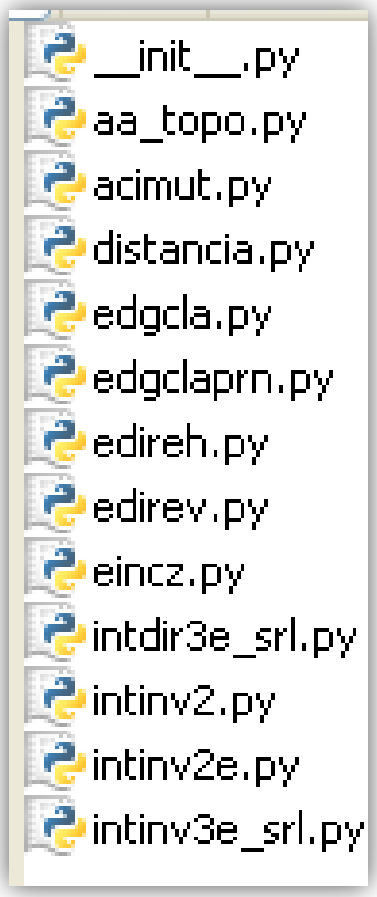
Por orden intuitivo desde el punto de vista de aplicación de las herramientas tenemos los siguientes paquetes:

- **general** que contiene funciones de tipo general y transversal al resto de herramientas.



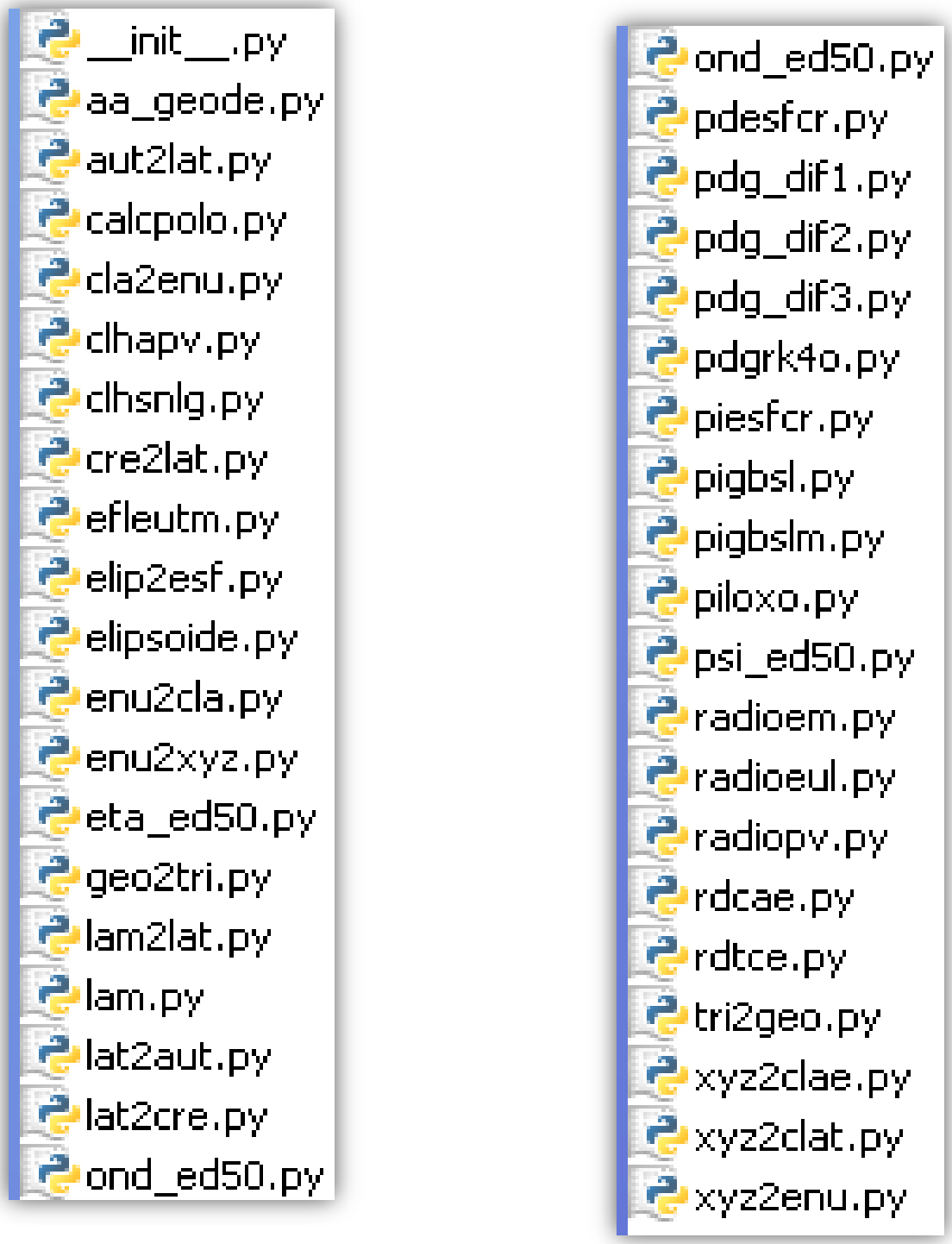
Módulos implementados dentro de general

- **topografia** que contiene aplicación topográficas



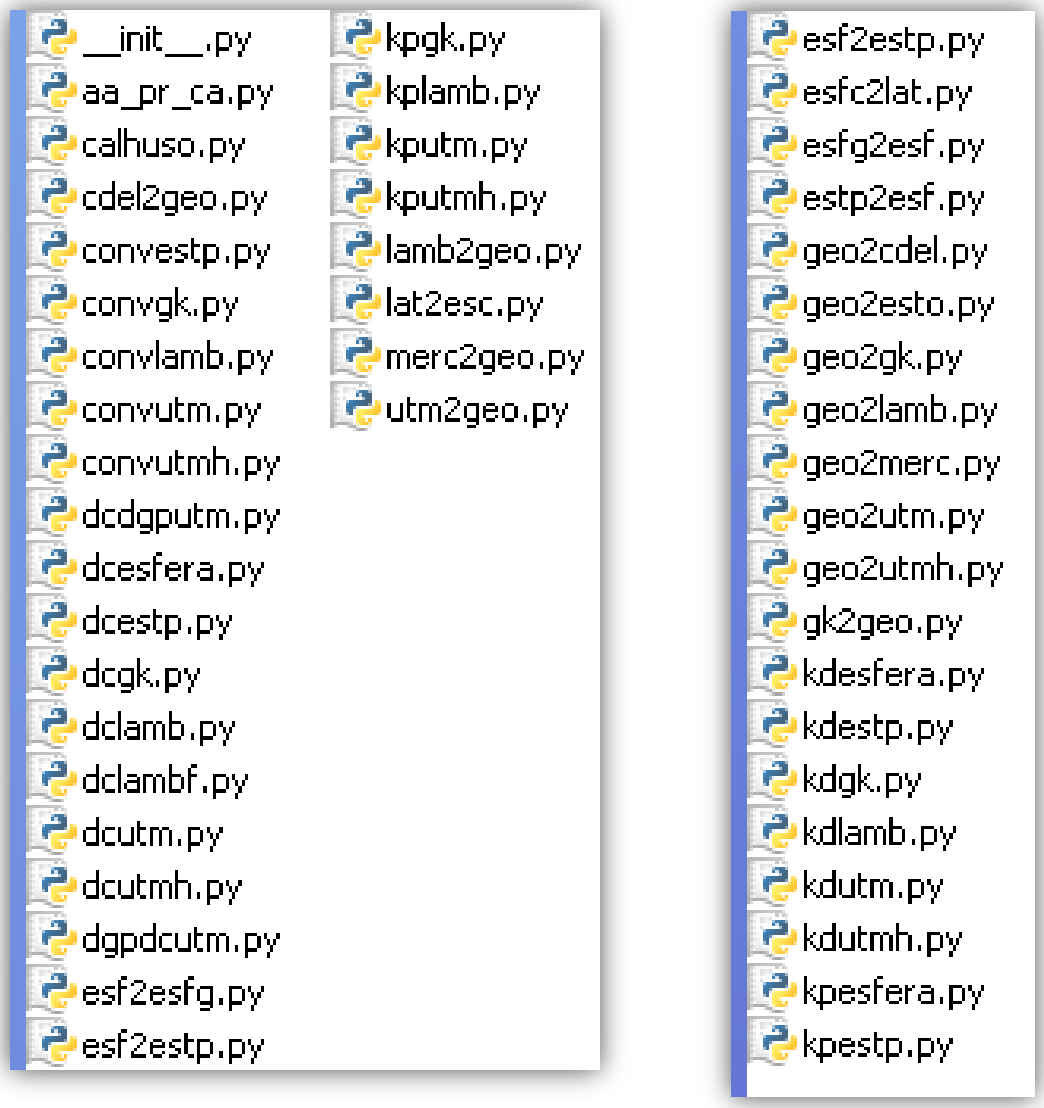
Módulos implementados dentro de topografia

- **geodesia.** Aplicaciones geodésicas



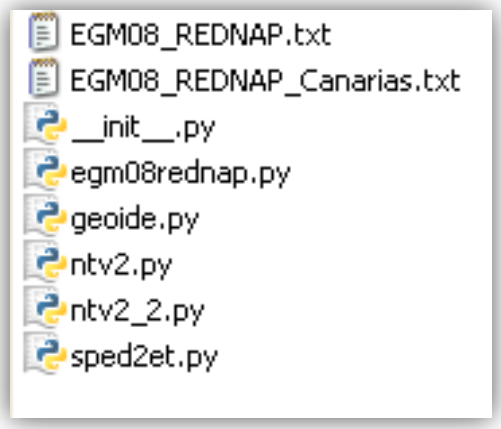
Módulos implementados dentro de geodesia

- **proy_cartograficas.** Herramientas de cálculos en diferentes proyecciones,etc.



Módulos implementados dentro de proy_cartograficas

- transformación. Aplicaciones con modelo de rejilla del IGN y EGM08REDNAP.



Módulos implementados dentro de transformación

- aplicaciones. Distintos ejemplos de aplicabilidad de las funciones y clases implementadas en los módulos de la librería.

Cada uno de estos paquetes (carpetas), como se ha podido ver, contiene a su vez varios módulos y en cada uno de ellos y con su mismo nombre se define una única función por módulo salvo en la carpeta de transformción donde como se ha comentado antes, nos adentramos dentro de la programación orientada a objetos y en algunos módulos se implementan clases y dentro de estas funciones, etc.

De esta manera al importar una función, la tenemos que importar desde el modulo que la contenga. Dentro de cada carpeta se aprecia la existencia de un fichero llamado __init__.py que convierte dicha carpeta en un paquete de Python.

Se advierte que desde el cierre de esta memoria, versión en papel, a la entrega total de documentación puede haberse ampliado el número de módulos de la librería "geotop" que sí se incluiría en el CD con la implementación de códigos.

4.2 DOCUMENTADO DE FUNCIONES

Se puede documentar una función en Python proporcionando una cadena de documentación (comentarios).

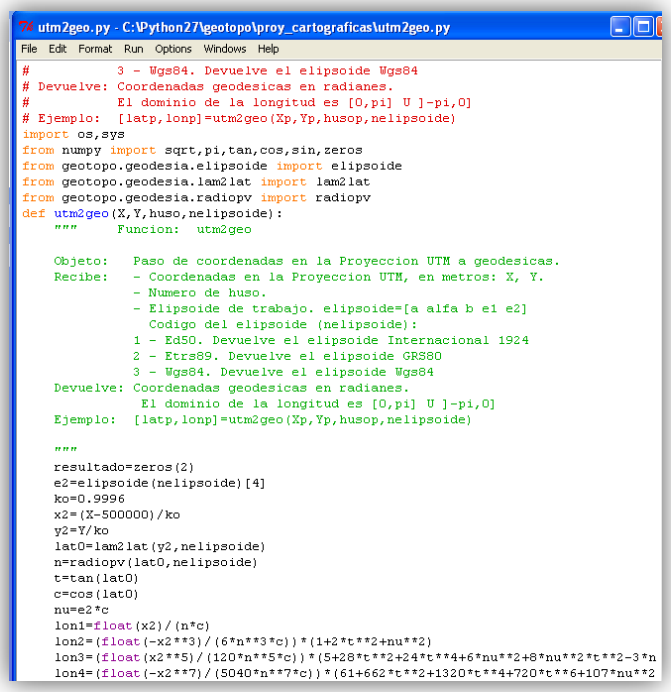
```
def buildConnectionString(params):
    """Crea una cadena de conexión partiendo de un diccionario de parámetros.

    Devuelve una cadena."""
```

Las comillas triples implican una cadena multilínea. Todo lo que haya entre el principio y el final de las comillas es parte de una sola cadena, incluyendo los retornos de carro y otras comillas. Pueden usarse para definir cualquier cadena, pero donde suelen estar más presentes es haciendo de cadena de documentación.

Todo lo que hay entre las comillas triples es la cadena de documentación de la función, y se usa para explicar lo que hace la función. En caso de que exista una cadena de documentación, debe ser la primera cosa definida en una función (esto es, lo primero tras los dos puntos). Técnicamente, no es necesario dotar a una función de una cadena de documentación pero debemos de hacerlo siempre. En Python, esto tiene un incentivo añadido: la cadena de documentación está disponible en tiempo de ejecución como atributo de la función.

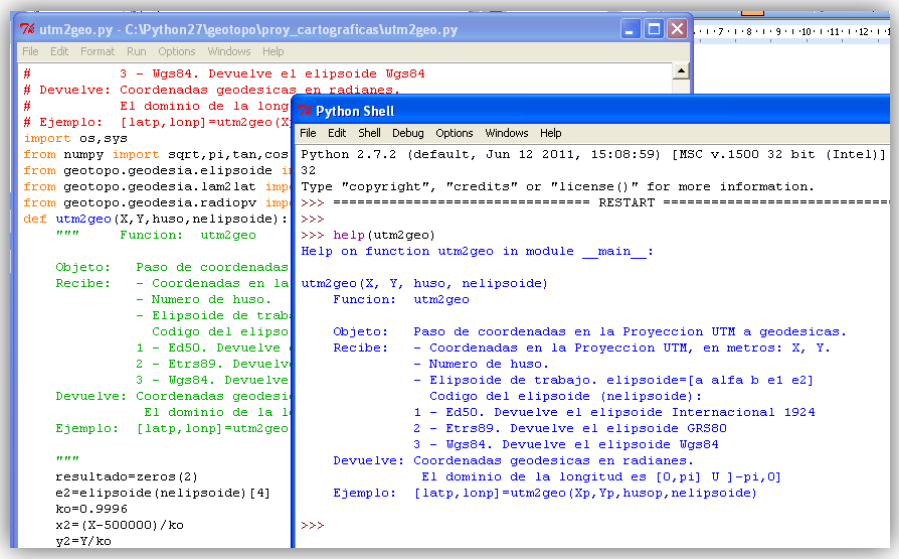
Lecturas complementarias sobre las funciones de documentación

- PEP 257 define las convenciones al respecto de las cadenas de documentación.
- La Guía de estilo de Python indica la manera de escribir una buena cadena de documentación.
- El Tutorial de Python expone convenciones para el espaciado dentro de las cadenas de documentación.



Ejemplo de documentado de funciones con cadena de documentación

Toda la información incluida dentro de una cadena de documentación de triple comillas está accesible en tiempo de ejecución. Así, si pedimos ayuda de una función, nos aparece toda la información incluida en dicha cadena de documentación.



Ejemplo de consulta de la ayuda de las funciones

En la siguiente web, podemos encontrar más información sobre el documentado de funciones en Python.
http://coderwar.com/2011/06/hola-python-la-utilidad-de-los-comentarios/

Además de los comentarios de las funciones, en Python, como en otros lenguajes de programación podemos incluir líneas o simples palabras de comentarios. En Python se usa el símbolo # para preceder los comentarios.
Podemos comentar las líneas de nuestro trabajo.

```
ganador='eres tú' #la variable ganador almacena un texto
```

Comentar un grupo de líneas.

```
#Lo siguiente es una función que estudiaremos posteriormente
def ganador( ):
    print 'quien más que tú'
```

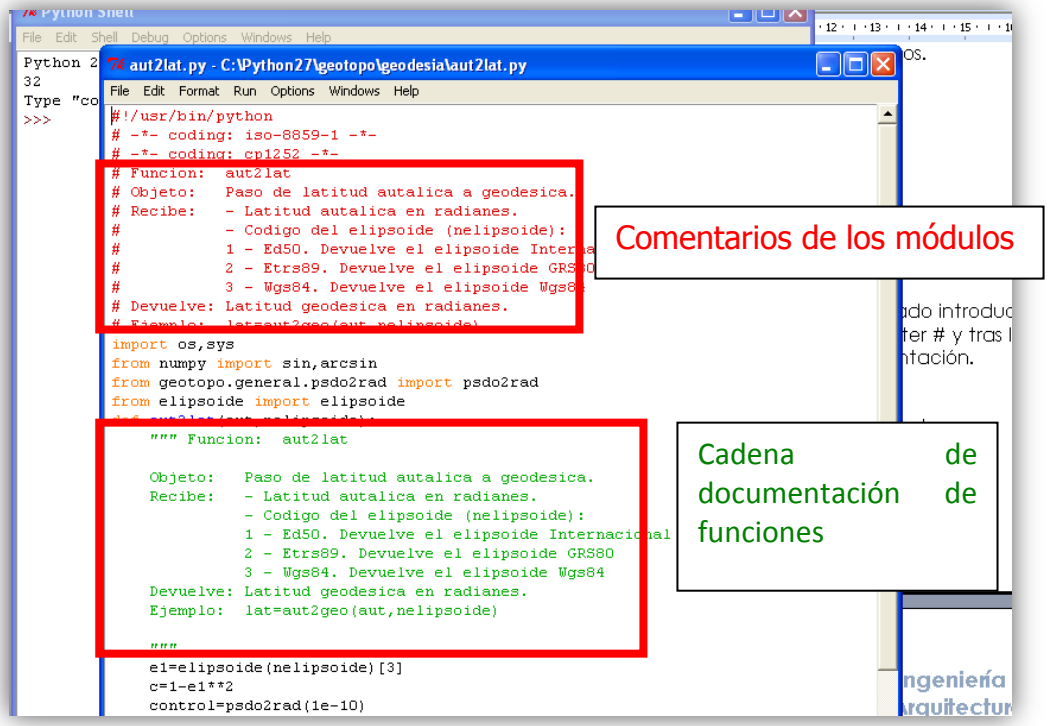
Determinar el tipo de codificación. Esto es importante para poder usar caracteres especiales como la tilde o la ñ. Generalmente se coloca en la primera línea del programa. Existen muchas codificaciones

```
# -*- coding: utf-8 -*-
```

Poner cabecera a nuestro programa. De esta forma, identificaremos el nombre del programador, la fecha en que se realizó y demás datos.

```
# Archivo: myprograma.py
# Programador: mi nombre y apellido
# Fecha: 08 Junio 2025
# Descripción:
#                    Mi programa donde
#                    resolveré un ejercicio
```

En el caso que nos atañe de la librería geotopo, hemos considerado introducir unos comentarios descriptivos de los módulos mediante el carácter # y tras la definición de las funciones se incluyen unas cadenas de documentación:



Ejemplo de la documentación de módulos y funciones...

Para evitar problemas de codificación, como ha ocurrido durante la implementación del código, se han incluido al inicio del mismo dos líneas para solucionar el problema. Estas líneas son:

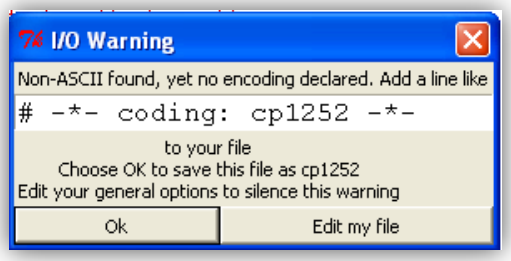
```
# -*- coding: iso 8859-1 *-*
```

ISO 8859-1 es una norma de la ISO que define la codificación del alfabeto latino, incluyendo diacríticos como letras acéntuadas, ñ,) y letras especiales, necesarios para la escritura de las siguientes lenguas originarias de Europa occidental: alemán, aragonés, asturiano, castellano, catalán, danés, escocés, español, feroés, finés, francés, gaélico, gallego, inglés, islandés, italiano, neerlandés, noruego, portugués, sueco y Euskera.

También conocida como Alfabeto Latino n.º 1 o ISO Latín 1.
Esta norma pertenece al grupo de juegos de caracteres de la ISO conocidos como ISO/IEC 8859 que se caracterizan por poseer la codificación ASCII en su rango inicial (128 caracteres) y otros 128 caracteres para cada codificación, con lo que en total utilizan 8 bits.

```
# -*- coding cp1252 *-*
```

Si trabajamos con IDLE, con el uso de los acentos (en general con los caracteres no ASCII) nos dará error. Si en el código de un programa escribimos una cadena que contenga caracteres no ASCII(acentos, ñ,¿,i), al guardar el archivo por primera vez, IDLE nos muestra el siguiente mensaje de aviso:



Si pulsamos el botón "Edit my file". Al principio del programa se insertará la línea siguiente

```
# -*- coding: cp1252 -*-
```

Que indica el juego de caracteres utilizado en el archivo. (cp1252 es el juego de caracteres de Windows para Europa occidental). A partir de entonces podemos guardar el archivo sin problemas.

Python no tiene una primera o última linea exigida pero sí que es habitual dar la localización de Python como primera línea en forma de comentario.
`#!/usr/bin/python.`

4.3 FUNCIONES IMPLEMENTADAS

A continuación se incluye una relación de las funciones implementadas por paquetes. Además de su nombre y un pequeño resumen de su aplicación se han incluido 3 apartados más:

En verde:

Datos de entrada. Indica los datos que necesita la función.

Datos de salida. Nos muestra si el resultado es único o una colección de resultados (siempre en forma vector en el orden indicado)

En rojo:

Llamadas o "importación" de módulos predefinidos del sistema: os,sys.

Llamadas a funciones de predefinidas del módulo numpy.

En morado:

Llamadas a funciones definidas en esta librería (geotopo), bien incluidas en el mismo subpaquete que la propia función definida o incluida en otro subpaquete.

De esta manera podemos tener una idea de las funciones implementadas, los datos de partida necesarios y los resultados a obtener, así como la interrelación de cada función con el resto de funciones de la librería geotopo.

Para el caso de cambio de proyección o de Sistema de Referencia Coordenado se ha tomado la nomenclatura "desde" "a" "ahora". Para señalar el "a" se ha elegido el número 2 por su pronunciación to (a). Así, por ejemplo para el paso de latitud geodésica a creciente, en nombre de la función será **lat2cre.**

4.3.1 Librería en Python "geotopo.general"

Autor: Dr. D. David Hernandez Lopez.
david.hernandez@ulcm.es
Migración a Python: Juan Pedro García Tendero
juanpedrogarcia@terra.es

Version: 1.0 para Python 2.7.

Última Modificación: Agosto 2011

areacor. - Cálculo de la superficie interior de un poligono sobre el plano

```
entrada=(matriz) 2columnas x,y
salida=superficie
import os,sys
from numpy import shape
```

busca2col. - Localiza la primera fila con los valores recibidos en las dos columnas indicadas.

Recibe: Valor en la primera y segunda columnas, posiciones de las columnas y matriz de busqueda.
Devuelve: Primera fila que cumple, comenzando en 0.

```
entrada=(valorcol1,poscol1,valorcol2,poscol2,matriz)
salida=orden de la primera fila que cumple, comenzando en 0.(ó -1 si no existe)
import os,sys
from numpy import shape
```

busca. - Localiza la primera ocurrencia de un número en la primera columna de una matriz

```
entrada=(valor,matriz)
salida=posición, número de fila comenzando en 0 (ó -1 si no existe)
import os,sys
from numpy import shape
```

buscacol. - Localiza la primera ocurrencia de un número en la fila indicada de una matriz

```
entrada=(valor,matriz,nfil)
salida=posición, número de columna, comenzando en 0 (ó -1 si no existe)
import os,sys
from numpy import shape
```

buscafil. - Localiza la primera ocurrencia de un número en la columna indicada de una matriz

```
entrada=(valor,matriz,ncol)
salida=posición, número de fila, comenzando en 0 (ó -1 si no existe)
import os,sys
from numpy import shape
```

ordena. - Ordena las filas de una matriz de menor a mayor según el contenido de la columna indicada

```
entrada=(colum,matriz)
salida=matriz de entrada ordenada
import os,sys
from numpy import shape
```

p_entera. - Da la parte entera de un número.

```
entrada=(número)
salida=parte entera del número
import os,sys
from numpy import floor
```

perimetro. - Calculo del perimetro de un polígono sobre el plano a partir de las coord de sus vértices.

```
entrada=(matriz ). 2columnas x,y
salida=perímetro
import os,sys
from numpy import sqrt,shape
```

psdo2rad. - Cambio de formato de ángulos.
Pasa del conocido como formato pseudo decimal sexagesimal
(ej: 40.24305678 son 40 g. 24 m. 30.5678 seg. sex.)
a radianes

```
entrada=(pseudesexa)
salida=radianes
import os,sys
from numpy import pi
from geotopo.general.p_entera import p_entera
```

rad2psdo. - Cambio de formato de ángulos.
Pasa un angulo de radianes al conocido como formato
pseudo decimal sexagesimal:
(ej: 40.24305678 son 40 g. 24 m. 30.5678 seg. sex.)

```
entrada=( radianes)
salida= pseudesexa
import os,sys
from numpy import pi
from geotopo.general.p_entera import p_entera
```

4.3.2 Librería en Python "geotopo.topografia"

Autor: Dr. D. David Hernandez Lopez.
david.hernandez@ulcm.es
Migración a Python: Juan Pedro García Tendero
juanpedrogarcia@terra.es

Version: 1.0 para Python 2.7.

Última Modificación: Agosto 2011

acimut. - Cálculo del acimut entre dos puntos:

```
entrada=(xi,yi,xj,yj)
salida=acimut en radianes
import os,sys
from numpy import arctan2,pi
```

distancia. – Cálculo de la distancia entre dos puntos del mismo

```
entrada=(xi,yi,xj,yj)
salida=distancia (misma unidad que coordenadas de entrada
import os,sys
from numpy import sqrt
```

edgcla. – Cálculo del error de una distancia geométrica utilizando la formulación clásica de teoría de errores

```
entrada=(dg,Instr.,ni,alt,nl).
  Dg=distancia geometrica
  Instr=Matriz de instrumentos
  Ni=Altura de instrumento
  Altp=Altura de prisma
  Nl=Número de veces de medida de distancia.
salida=error de distancia en metros.
import os,sys
from numpy import sqrt,pi
```

edgclaprn. – Calculo del error de una distancia geometrica utilizando la formulacion clásica de teoría de errores. Impresión de resultados

```
entrada=(dg,Instr.,ni,alt,nl,fsalida).
  dg=distancia geometrica
  instr=Matriz de instrumentos
  ni=Altura de instrumento
  altp=Altura de prisma
  nl=Número de veces de medida de distancia.
```

```
  fsalida=Fichero de salidade datos
salida=error de distancia en metros.
import os,sys
from numpy import sqrt,pi
```

edireh. – Cálculo del error de la lectura horizontal realizada con un teodolito, utilizando la formulación clásica de teoría de errores.

```
entrada=(Instr.,ni,dis,altp,nl).
salida=error de lectura horizontal en radianes.
import os,sys
from numpy import sqrt,pi
```

edirev. – Cálculo del error de la lectura vertical realizada con un teodolito, utilizando la formulación clásica de teoría de errores.

```
entrada=(instr.,ni, nl).
salida=error de lectura cenital en radianes en radianes.
import os,sys
from numpy import sqrt,pi
```

eincz. – Cálculo del error de un desnivel trigonométrico, utilizando la formulación clásica de teoría de errores.

```
entrada=(dg,lv,ap,edg,elv,Instr.,ni).
salida=xp,yp,eplani,solz,esolz
import os,sys
from numpy import pi,sqrt,cos,tan
```

intinv2. – Cálculo de la intersección inversa simple en el plano.

```
entrada=(datos,instru).
salida= xp,yp,angmin
import os,sys
from numpy import sin,cos,tan,arctan2,pi,zeros
from geotopo.general.ordena import ordena
```

intinv2e. – Cálculo de la intersección inversa simple en el plano. Realiza también la previsión de error de la posición calculada a partir de las características del instrumental, etc.

```
entrada=(datos,instru).
salida= xp,yp,eplani
import os,sys
from numpy import shape,pi,sin,cos,tan,arctan2,dot,transpose,zeros,sqrt
from geotopo.topografia.edireh import edireh
```

intinv3e. – Cálculo de la intersección inversa 3D simple. Realiza también la previsión de error de la posición calculada a partir de las características del instrumental, etc.

```
entrada=(datos,instru).
salida= xp,yp,eplani,vzp,vezp
import os,sys
from numpy import zeros,shape,sin,cos
from geotopo.topografia.intinv2e import intinv2e
from geotopo.topografia.distancia import distancia
from geotopo.topografia.edirev import edirev
from geotopo.topografia.edgcla import edgcla
from geotopo.topografia.eincz import eincz
```

4.3.3 Librería en Python "geotopo.geodesia"

Autor: Dr. D. David Hernandez Lopez.
david.hernandez@ulcm.es
Migración a Python: Juan Pedro García Tendero
juanpedrogarcia@terra.es

Versión: 1.0 para Python 2.7.

Última Modificación: Agosto 2011

aut2lat. - paso de latitud autálica a geodésica.

```
entrada=(aut,nelipsoide)
salida=lat
import os,sys
from numpy import sin,arcsin
from geotopo.general.psdo2rad import psdo2rad
from geotopo.general.psdo2rad import psdo2rad
```

calcpolo. - cálculo del polo de la esfera, a partir de tres puntos no alineados de la misma, de forma que pasen a estar en un mismo paralelo tras realizarse el cambio de polo.

```
entrada(lat1,lon1,lat2,lon2,lat3,lon3)
salida=[latnp,lonnp,colat]
import os,sys
from numpy import pi,sin,cos,arccos,arctan,arctan2,zeros
```

cla2enu. - paso de un vector geodésico, expresado en componentes polares (observables clásicos), a componentes ENU.

```
entrada(az,v,dg)
salida=[ie,in,iu]
import os,sys
from numpy import sin,cos,zeros
```

clhapv. - lectura horizontal corregida por altitud del punto visado.

```
entrada=(lhz,az,lat1,lat2,h2,nelipsoide)
salida=lhz12c
import os,sys
from numpy import sin,cos,pi
from elipsoide import elipsoide
from radioem import radioem
```

clhsnlg. - lectura horizontal corregida por paso de la sección normal a la línea geodésica.

```
entrada=(lhz,s12,az12,lat1,lat2,nelipsoide)
salida=lhz12c
import os,sys
from numpy import sin,cos,pi
from elipsoide import elipsoide
from radioem import radioem
```

cre2lat. - paso de latitud creciente a geodésica.

```
entrada=(cre,nelipsoide)
salida=lat
import os,sys
from numpy import sin,cos,tan,arctan,exp,pi
from geotopo.general.psdo2rad import psdo2rad
from elipsoide import elipsoide
```

efleutm. - paso de la elipse de error de un punto del plano tangente al elipsoide al plano de la proyección UTM.

```
entrada=(lat,lon,semiae,semibe,aziae,nelipsoide,huso)
salida=[semia,simib,azia]
import os,sys
from numpy import zeros,pi
from geotopo.proy_cartograficas.kputm import kputm
from geotopo.proy_cartograficas.kputmh import kputmh
from geotopo.proy_cartograficas.convutm import convutm
from geotopo.proy_cartograficas.convutmh import convutmh
```

elip2esf. - determinación de la esfera tangente al elipsoide en un paralelo indicado.

```
entrada=(nelipsoide,lat)
salida=[a,f,b,e1,e2]esfera
import os,sys
from numpy import zeros
from elipsoide import elipsoide
from radiopv import radiopv
```

elipsoide - calculo los 5 parámetros (a,f,b,e1 y 22) de los elipsoides ED50, GRS80 y WGS84 en función de un código 1,2,3, respectivamente

```
entrada=(nelipsoide) 1, 2 ó 3 según ED50,GRS80 ó WGS84 respectivamente
salida=[a,f,b,e1,e2]
import os,sys
from numpy import sqrt, zeros
```

enu2cla. - paso de un vector geodésico, expresado en componentes ENU, a componentes polares (observables clásicos).

```
entrada=ie,in,iu
salida=[az,dg,v]
import os,sys
from numpy import zeros,arcsen,arctan2,pi,sqrt
```

enu2xyz. - paso de un vector geodésico, expresado en componentes ENU, a componentes cartesianas tridimensionales.

```
entrada=(lat,lon,ie,in,iu)
salida=[ix,iy,iz]
import os,sys
from numpy import sin,cos,zeros
```

eta_ed50. - componente de la desviación relativa de la vertical en Ed50 según la dirección del primer vertical.

```
entrada=(lat,lon)
salida= eta_ed50
import os,sys
from numpy import pi,sqrt,cos,sin,tan
from elipsoide import elipsoide
from radiopv import radiopv
from radioem import radioem
```

geo2tri. - paso de coordenadas geodésicas a cartesianas tridimensionales, para un punto y elipsoide determinados.

```
entrada=(lat,lon,h,nelipsoide
salida=[X,Y,Z]
import os,sys
from numpy import cos,sin,zeros
from elipsoide import elipsoide
from radiopv import radiopv
```

lam2lat. - cálculo de la latitud geodésica que le corresponde a una determinada longitud de arco de meridiano en el elipsoide indicado.

```
entrada=(lam,nelipsoide
salida=lat
import os,sys
from numpy import cos,sin
from elipsoide import elipsoide
from geotopo.general.rad2psdo import rad2psdo
```

lam. - cálculo de la longitud de arco de meridiano a partir de la latitud geodésica y del elipsoide.

```
entrada=(lam,nelipsoide
salida=lam
import os,sys
from numpy import sin cos
from elipsoide import elipsoide
```

lat2aut. - paso de latitud geodésica a autálica.

```
entrada=(lat,nelipsoide)
salida=aut
import os,sys
from numpy import sin,log
from elipsoide import elipsoide
```

lat2cre. - paso de latitud geodésica a creciente.

```
entrada=(lat,nelipsoide)
salide=cre
import os,sys
from numpy import sin,tan,pi,log
```

ond_ed50. - cálculo de la ondulación del geoide de un punto en Ed50.

```
entrada =(lat,lon)
salida=ondulación
```

```
import os,sys
from numpy import pi,sqrt,cos,sin,tan
```

pdesfcr. - problema directo en la esfera con el método de cambio de base.

```
entrada=(lat1,lon1,s,azi,resfera)
salida=[lat,lon]
import os,sys
from numpy import cos,sin,tan,arcsin,arcos,arctan2,pi,zeros,transpose,dot
```

pdg_dif1. - problema directo de la geodesia con la primera derivada.

```
entrada=(lat1,lon1,s,a12,nelipsoide)
salida=[lat2,lon2,azi,21]
import os,sys
from numpy import zeros,sin,cos,tan,pi
from radiopv import radiopv
from radioem import radioem
```

pdg_dif2. - problema directo de la geodesia con la primera y segunda derivada.

```
entrada=(lat1,lon1,s,a12,nelipsoide)
salida=[lat2,lon2,azi,21]
import os,sys
from numpy import zeros,sin,cos,tan,pi
from radiopv import radiopv
from radioem import radioem
from elipsoide import elipsoide
```

pdg_dif3. - problema directo de la geodesia con la primera, segunda y tercera derivada.

```
entrada=(lat1,lon1,s,a12,nelipsoide)
salida=[lat2,lon2,azi,21]
import os,sys
from numpy import zeros,sin,cos,tan,pi
from radiopv import radiopv
from radioem import radioem
from elipsoide import elipsoide
```

pdgrk4o. - problema directo de la geodesia según método de integración numérica de Runge-Kutta de cuarto orden.

```
entrada=(lat1,lon1,s,a12,nelipsoide)
salida=[lat2,lon2
import os,sys
from numpy import zeros,sin,cos,tan,pi,sqrt
from radiopv import radiopv
```

```
from radioem import radioem
```

piesfcr. - problema inverso en la esfera con el método de cambio de base.

```
entrada=( lat1,lon1,lat2,lon2,resfera)
salida=[s12,a12]
import  os,sys
from numpy import cos,sin,tan,arcsen,arccos,arctan2,dot,zeros,pi
```

pigbsl. - problema inverso de la geodesia según el método de Bessel.

```
entrada=( lat1,lon1,lat2,lon2,nelipsoide)
salida=[ s12,a12,a21]
import os,sys
from numpy import zeros,sin,cos,tan,arcsin,arctan,arctan2,arccos,sqrt,pi
```

pigbslm. - problema inverso de la geodesia según el método de Bessel modificado por D. David Henández López

```
entrada=( lat1,lon1,lat2,lon2,nelipsoide)
salida=[ s12,a12,a21]
import os,sys
from numpy import cos,sin,tan,arcsin,arcos,arctan,arctan2,zeros,sqrt,pi
from elipsoide import elipsoide
from radiopv import radiopv
from piesfcr import piesfcr
from pdesfcr import pdesfcr
```

piloxo. - problema inverso para la curva loxodrómica.

```
entrada=(lat1,lon1,lat2,lon2,nelipsoide)
salida=[dl,a12]
import os,sys
from numpy import arctan2,cos,zeros,pi
from elipsoide import elipsoide
from lat2cre import lat2cre
from radiopv import radiopv
from lam import lam
```

psi_ed50. - componente de la desviación relativa de la vertical en Ed50 según la dirección del meridiano.

```
entrada=(lat,lon)
salida=psi_ed50
import os,sys
from numpy import pi,sqrt,cos,sin,tan
from radioem import radioem
from radiopv import radiopv
```

radioem. - radio de curvatura de la elipse meridiana, en un punto y elipsoide determinados.

```
entrada=(lat,nelipsoide)
salida=radioem
import os,sys
from numpy import sin
from elipsoide import elipsoide
```

radioeul. - radio de la sección normal de un determinado acimut, a partir del teorema de Euler.

```
entrada=(lat,az,nelipsoide)
salida=radieul
import os,sys
from numpy import sin,cos
from radioem import radioem
from radiopv import radiopv
```

radiopv. - radio de curvatura del primer vertical, en un punto y elipsoide determinados.

```
entrada=(lat,nelipsoide)
salida=radiopv
import os,sys
from numpy impor sqrt,sin
from elipsoide import elipsoide
```

rdcae. - reducción de la distancia de la cuerda al arco elipsoide.

```
entrada=( dc,az,lat1,nelipsoide)
salida=recae
impor os,sys
from numpy import sin,cos,arcsen
from radioem import radioem
from radiopv import radiopv
```

rdtce. - reducción de la distancia del terreno a la cuerda elipsoide.

```
entrada=(dg,az,lat1,h1,h2,nelipsoide)
salida=rdtce
import os,sys
from numpy import sin,cos,sqrt
from radioem import radioem
from radiopv import radiopv
```

tri2geo. - paso de coordenadas cartesianas tridimensionales a geodésicas, en un punto y elipsoide determinados.

```
entrada=(X,Y,Z,nelipsoide)
salida=[lat,lon,h]
import os,sys
from numpy import cos,sin,tan,arctan,arctan2,sqrt,zeros
from elipsoide import elipsoide
from radiopv import radiopv
```

xyz2clae. - Paso de un vector geodésico, expresado en componentes cartesianas tridimensionales, a observables clásicos sobre el elipsoide y desnivel elipsoidal.

```
entrada=( X1,Y1,Z1,DX,DY,DZ,nelipsoide)
salida=[s12,az12,h12]
import os,sys
from numpy import zeros
from tri2geo import tri2geo
from pigbslm import pigbslm
```

xyz2clat. - Paso de un vector geodésico, expresado en componentes cartesianas tridimensionales, a componentes polares en el terreno.

```
entrada=( lat1,lon1,ix,iy,iz)
salida=[ dg,az12,lv]
import os,sys
from numpy import zeros
from xyz2enu import xyz2enu
from enu2cla import enu2cla
```

xyz2enu. - Paso de un vector geodésico, expresado en componentes cartesianas tridimensionales, a componentes ENU.

```
entrada=(lat,lon,IX,IY,IZ)
salida=[ie,in,iu]
import os,sys
from numpy import sin,cos,zeros.
```

4.3.4 Librería en Python "geotopo.proy_cartograficas"

Autor: Dr. D. David Hernandez Lopez.
david.hernandez@ulcm.es
Migración a Python: Juan Pedro García Tendero
juanpedrogarcia@terra.es

Versión: 1.0 para Python 2.7.

Última Modificación: Agosto 2011

calhuso. - cálculo del huso correspondiente a un punto en la proyección UTM a partir de su longitud geodésica.

```
entrada=(lon)
salida=huso
import os,sys
from numpy import
```

cdel2geo. - paso de un punto, de la proyección cilíndrica directa equivalente de Lambert, al elipsoide.

```
entrada=(X,Y,Lon0,TX,TY,nelipsoide)
salida=[lat,lon]
import os,sys
from numpy import zeros,arctan2,pi
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.aut2lat import aut2lat
```

convestp. - cálculo de la convergencia de meridianos de un punto de la esfera, en la proyección estereográfica polar.

```
entrada=(lon,lon0)
salida=convest
import os,sys
```

convgk. - cálculo de la convergencia de meridianos de un punto del elipsoide, en la proyección general de Gauss-Krüger.

```
entrada=(lat,lon,lon0,nelipsoide)
salida=convgk
import os,sys
from numpy import sin,cos,tan
from geotopo.geodesia.elipsoide import elipsoide
```

convlamb. - cálculo de la convergencia de meridianos de un punto del elipsoide, en la proyección general cónica conforme de Lambert.

```
entrada=(lon,lat0,lon0)
salida=convlamb
import os,sys
from numpy import sin
```

convutm. - cálculo de la convergencia de meridianos de un punto del elipsoide, en la proyección UTM, calculada en el huso que le corresponde al punto por su longitud geodésica.

```
entrada=(lat,lon,nelipsoide)
salida=convutm
import os,sys
from numpy import tan,cos,sin,pi
from cahuso import calhuso
from geotopo.geodesia.elipsoide import elipsoide
```

convutmh. - cálculo de la convergencia de meridianos de un punto del elipsoide, en la proyección UTM, calculada en el huso indicado.

```
entrada=(lat,lon,nelipsoide)
salida=convutmh
import os,sys
from numpy import tan,cos,sin,pi
from geotopo.geodesia.elipsoide import elipsoide
```

dcdgputm. - paso de la distancia cuerda en la proyección UTM, entre dos puntos, a la correspondiente a la proyección UTM de la línea geódesica, que los une en el elipsoide.

```
entrada=(lat1,lon,1,lat2,lon2,nelipsoide
salida=corrección en metros
import os,sys
from numpy import cos
from utm2geo import utm2geo
from geo2utmh import geo2utmh
from calhuso import calhuso
from getopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.radioem import radioem
from geotopo.topografia.acimut import acimut
from geotopo.topografia.distancia import distancia
```

dcesfera. - cálculo de la reducción angular de la cuerda, en el extremo inicial de una geodésica en el elipsoide, al pasar a la proyección general conforme en una esfera.

```
entrada=(lat1,lon1,lat2,lon2,lat0,nelipsoide)
salida=dc (reducción angular de la cuerda en radianes
import os,sys
from numpy import sin,cos
from lat2esc import lat2esc
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.pigbslm import pigbslm
from geotopo.geodesia.pdgrk4o import pdgrk4o
```

dcestp. - cálculo de la reducción angular de la cuerda, en el extremo inicial de una geodésica en la esfera, al pasar a la proyección estereográfica polar.

```
entrada=(lat1,lon1,lat2,lon2,lat0,lon0,p,nelipsoide)
salida=dc(reducción angular de la cuerda en radianes)
import os,sys
from numpy import cos,sin
from kplamb import kplamb
from geotopo.geodesia.pigbslm import pigbslm
from geotopo.geodesia.pdgrk4o import pdgrk4o
from geotopo.geodesia.radiopv import radiopv
```

dcgk. - cálculo de la reducción angular de la cuerda, en el extremo inicial de una geodésica en el elipsoide, al pasar a la proyección general de Gauss-Krüger.

```
entrada=(lat1,lon1,lat2,lon2,lon0,p,nelipsoide)
salida= dc(reducción angular de la cuerda en radianes)
import os,sys
from numpy import cos
from geo2gk import geo2gk
from gk2geo import gk2geo
from geotopo.geodesia.radioem import radioem
from geotopo.geodesia.radiopv import radiopv
from getopo.geodesia.elipsoide import elipsoide
```

dclamb. - cálculo de la reducción angular de la cuerda, en el extremo inicial de una geodésica en el elipsoide, al pasar a la proyección general cónica conforme de Lambert.

```
entrada=(lat1,lon1,lat2,lon2,lat0,p,elipsoide
salida= dc(reducción angular de la cuerda en radianes)
import os,sys
from numpy import sin,tan,cos,exp,zeros
from geo2lamb import geo2lamb
from kplamb import kplamb
```

```
from kdlamb import kdlamb
from geotopo.topografia,acimut import acimut
from geotopo.topografia.distancia import distancia
from geotopo.geodesia.radiopv import radiopv
```

dclambf. - cálculo de la reducción angular de la cuerda, en el extremo inicial de una geodésica en el elipsoide, al pasar a la proyección general cónica conforme de Lambert, con más precisión que la función anterior.

```
entrada=(lat1,lon1,lat2,lon2,lat0,lon0,p,nelipsoide)
salida= dc(reducción angular de la cuerda en radianes)
import os,sys
from numpy import cos, sin
from kplamb import kplamb
from geotopo.geodesia.pigbslm import pigbslm
from geotopo.geodesia.pdgrk4o import pdgrk4o
from geotopo.geodesia.radiopv import radiopv
```

dcutm. - cálculo de la reducción angular de la cuerda, en el extremo inicial de una geodésica en el elipsoide, al pasar a la proyección UTM, calculada en el huso que le corresponde al punto por su longitud geodésica.

```
entrada=( lat1,lon1,lat2,lon2,nelipsoide)
salida= dc(reducción angular de la cuerda en radianes
import os,sys
from numpy import cos,pi
from geo2utmh import geo2utmh
from utm2geo import utm2geo
from calhuso import calhuso
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.radioem import radioem
```

dcutmh. - cálculo de la reducción angular de la cuerda, en el extremo inicial de una geodésica en el elipsoide, al pasar a la proyección UTM, calculada en el huso indicado.

```
entrada=( lat1,lon1,lat2,lon2,huso,nelipsoide)
salida= dc(reducción angular de la cuerda en radianes
import os,sys
from numpy import cos,pi
from geo2utmh import geo2utmh
from utm2geo import utm2geo
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.radioem import radioem
```

dgpdcutm. - paso de la distancia geodésica proyectada, en el plano UTM, a la distancia correspondiente a la cuerda en esta proyección, para una geodésica del elipsoide.

```
entrada=( lat1,lon1,lat2,lon2,s,nelipsoide)
salida=corrección
import os,sys
from numpy import cos
from geo2utmh import geo2utmh
from utm2geo import utm2geo
from calhuso import calhuso
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.radioem import radioem
from geotopo.topografia,acimut import acimut
```

esf2esfg. - paso de un punto, de una esfera, a la esfera con cambio de polo. También se calcula el giro de los círculos máximos, motivado por el cambio de polo.

```
entrada=(lat,lon.latp.lonp)
salida=[lat,lon,g]
import os,sys
from numpy import cos,sin,tan,arcsen,arctan,dot,zeros,pi
from geotopo.geodesia.piesfcr import piesfcr
```

esf2estp. - paso de un punto de la esfera, a las planas de la proyección estereográfica polar.

```
entrada=(lat,lon.lon0,radio,tx,ty)
salida=[x,y] estp
import os,sys
from numpy import zeros,sin,cos
```

esfg2esf. - paso de un punto, de la esfera con cambio de polo, a la esfera original. También se calcula el giro de los círculos máximos, motivado por el cambio de polo.

```
entrada=(latn,lonn,latp,lonp
salida=[lat0,lon0,giro
import os,sys
from numpy import cos,sin,tan,arcsen,arctan2,dot,zeros,pi,transpose
from geotopo.geodesia.piesfcr import piesfcr
```

esfc2lat. - paso de latitud, sobre la esfera, a la correspondiente sobre el elipsoide, según la proyección general conforme sobre la esfera.

```
entrada=(lat,lat0,nelipsoide
```

```
salida=lat
import os,sys
from numpy import sin,tan,arctan,pi
from geotopo.general.psdo2rad import psdo2rad
from geotopo.geodesia.elipsoide import elipsoide
```

estp2esf. - paso de un punto, de la proyección estereográfica polar, a la esfera.

```
entrada=(x,y,lon0,radio,tx,ty)
salida=[x,y]
import os,sys
from numpy import zeros,arctan2,arctan,sin,pi
```

geo2cdel. - paso de un punto, del elipsoide, a la proyección cilíndrica directa equivalente de Lambert.

```
entrada=( lat,lon,lon0,tx,ty,nelipsoide)
salida=[X,Y]
import os,sys
from numpy import zeros
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.lat2aut import lat2aut
```

geo2esto. - paso de una nube de puntos del elipsoide a la proyección estereográfica oblicua óptima.

```
entrada=(lat,lon,lat0,lon0,tx,ty,nelipsoide)
salida=[X,Y]
import os,sys
from numpy import
from lat2esc import lat2esc
from esf2esfg import esf2esfg
from esf2estp import esf2estp
from geotopo.geodesia.radiopv import radiopv
```

geo2gk. - paso de un punto, del elipsoide, a la proyección general de Gauss-Krüger.

```
entrada=( lat,lon,lon0,p,tx,ty,nelipsoide)
salida= [x,y]gk
import os,sys
from numpy import cos,tan,pi,zeros
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.lam import lam
```

geo2lamb. - paso de un punto, del elipsoide, a la proyección general cónica conforme de Lambert.

```
entrada=( lat,lon,lat0,lon0,p,tx,ty,nelipsoide)
salida=[x,y]lambert
import os,sys
from numpy import sin,tan,cos,exp,zeros
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.lat2cre import lat2cre
```

geo2merc. - paso de un punto, del elipsoide, a la proyección de Mercator.

```
entrada=( lat,lon,lon0,tx,ty,nelipsoide)
salida= [x,y]mercator
import os,sys
from numpy import zeros
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.lat2cre import lat2cre
```

geo2utm. - paso de un punto, del elipsoide, a la proyección UTM, en el huso que le corresponde al punto por su longitud geodésica.

```
entrada=(lat,lon,nelipsoide)
salida=[X,Y] UTM
import os,sys
from numpy import zeros,tan,cos,pi
from calhuso import calhuso.
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.lam import lam
```

geo2utmh. - paso de un punto, del elipsoide, a la proyección UTM, en el huso indicado.

```
entrada=(lat,lon,huso,nelipsoide)
salida=[X,Y] UTM
import os,sys
from numpy import zeros,tan,cos,pi
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.lam import lam
```

gk2geo. - paso de un punto, de la proyección general de Gauss-Krüger, al elipsoide.

```
entrada=(X,Y,lon0,p,tx,ty,nelipsoide)
salida=[lat,lon]
import os,sys
```

```
from numpy import zeros,tan,cos,pi
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.lam2lat import lam2lat
from geotopo.geodesia.radiopv import radiopv
```

kdesfera. - cálculo del coeficiente de anamorfosis lineal, a aplicar a una geodésica del elipsoide, al proyectarla en la proyección general conforme sobre una esfera. Utiliza la integración numérica de Simpson.

```
entrada=(lat1,lon1,lat2,lon2,lat0,nelipsoide)
salida= kdesfera
import os,sys
from kpesfera import kpesfera
from geotopo.geodesia.pigbslm import pigbslm
from geotopo.geodesia.pdgrk4o import pdgrk4o
```

kdestp. - cálculo del coeficiente de anamorfosis lineal, a aplicar a una geodésica de la esfera, al proyectarla en la proyección estereográfica polar. Utiliza la integración numérica de Simpson.

```
entrada=( lat1,lon1,lat2,lon2,lon0,radio,tx,ty)
salida=kdestp
import os,sys
from kpestp import kpestp
from esf2estp import esf2estp
from estp2esf import estp2esf
```

kdgk. - cálculo del coeficiente de anamorfosis lineal, a aplicar a una geodésica del elipsoide, al proyectarla en la proyección general de Gauss-Krüger. Utiliza la integración numérica de Simpson.

```
entrada=( lat1,lon1,lat2,lon2,lon0,p,nelipsoide)
salida=kdgk
import os,sys
from geo2gk import geo2gk
from gk2geo import gk2geo
from kpgk import kpgk
```

kdlamb. - cálculo del coeficiente de anamorfosis lineal, a aplicar a una geodésica el elipsoide, al proyectarla en la proyección general cónica conforme de Lambert. Utiliza la integración numérica de Simpson.

```
entrada=( lat1,lon1,lat2,lon2,lat0,lon0,p,nelipsoide)
salida=kdlamb
import os,sys
from geolamb import geolamb
from lamb2geo import lamb2geo
from kplamb import kplamb
```

kdutm. - cálculo del coeficiente de anamorfosis lineal, a aplicar a una geodésica del elipsoide, al proyectarla en la proyección UTM, en el huso del primer punto. Utiliza la integración numérica de Simpson.

```
entrada=( lat1,lon1,lat2,lon2,nelipsoide)
salida=kdutm
import os,sys
from calhuso import calhuso.
from geo2utmh import geo2utmh
from utm2geo import utm2geo
from kputmh import kputmh
```

kdutmh. - cálculo del coeficiente de anamorfosis lineal, a aplicar a una geodésica del elipsoide, al proyectarla en la proyección UTM, en el huso indicado. Utiliza la integración numérica de Simpson.

```
entrada= lat1,lon1,lat2,lon2,huso,helipsoide)
salida=kdutmh
import os,sys
from geo2utmh import geo2utmh
from utm2geo import utm2geo
from kputmh import kputmh
```

kpesfera. - cálculo del coeficiente de anamorfosis lineal puntual, para un punto del elipsoide, en la proyección general conforme sobre una esfera.

```
entrada=( lat,lat0,nelipsoide)
salida=kpesfera
import os,sys
from numpy import cos,sin,tan,exp
from lat2esc import lat2esc
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.elipsoide import elipsoide
```

kpestp. - cálculo del coeficiente de anamorfosis lineal puntual, para un punto de la esfera, en la proyección estereográfica polar.

```
entrada=(lat)
salida=kpestp
import os,sys
from numpy import sin
```

kpgk. - cálculo del coeficiente de anamorfosis lineal puntual, para un punto del elipsoide, en la proyección general de Gauss-Krüger.

```
entrada=(lat,lon,lon0,p,nelipsoide)
salida=kpgk
```

```
import os,sys
from numpy import sin,cos,tan
from clahuso import calhuso
from geotopo.geodesia.elipsoide import elipsoide
```

kplamb. - cálculo del coeficiente de anamorfosis lineal puntual, para un punto del elipsoide, en la proyección general cónica conforme de Lambert.

```
entrada=(lat,lat0,p,nelipsoide)
salida=kplamb
import os,sys
from numpy import cos,sin,tan,exp
from geotopo.geodesia.lat2cre import lat2cre
from geotopo.geodesia.radiopv import radiopv
```

kputm. - cálculo del coeficiente de anamorfosis lineal puntual, para un punto del elipsoide, en la proyección UTM, en el huso que le corresponde al punto por su longitud geodésica.

```
entrada=(lat,lon,nelipsoide)
salida=kputm
import os,sys
from numpy import tan,cos,sin,pi
from calhuso import calhuso.
from geotopo.geodesia.elipsoide import elipsoide
```

kputmh. - cálculo del coeficiente de anamorfosis lineal puntual, para un punto del elipsoide, en la proyección UTM, en el huso indicado.

```
entrada=(lat,lon,huso,nelipsoide)
salida=kputmh
import os,sys
from numpy import sin,cos,tan,pi
from geotopo.geodesia,elipsoide import elipsoide
```

lamb2geo. - paso de un punto, de la proyección general cónica conforme de Lambert, al elipsoide.

```
entrada=( x,y,lat0,lon0,p,tx,ty,nelipsoide)
salida=[lat,lon]
import os,sys
from numpy import sin,tan,arctan,pi,log,sqrt,zeros
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.lat2cre import lat2cre
from geotopo.geodesia.cre2lat import cre2lat
```

lat2esc. - paso de la latitud geodésica, sobre el elipsoide, a la correspondiente sobre la esfera, según la proyección general conforme sobre la esfera.

```
entrada=(lat,lat0,nelipsoide)
salida=latesfc
import os,sys
from numpy import sin,tan,arctan,pi
from geotopo.geodesia.elipsoide import elipsoide
```

merc2geo. - paso de un punto, de la proyección de Mercator, al elipsoide.

```
entrada= (x,y,lon0,tx,ty,nelipsoide)
salida=[lat,lon]
import os,sys
from numpy import zeros,arctan2,pi
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.cre2lat import cre2lat
```

utm2geo. - paso de un punto, de la proyección UTM, al elipsoide.

```
entrada=(X,Y,huso,nelipsoide)
salida=[lat,lon]
import os,sys
from numpy import sqrt,pi,tan,cos,sin,zeros
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.lam2lat import lam2lat
from geotopo.geodesia.radiopv import radiopv
```

4.3.5 Librería en Python "geotopo.transformacion"

Autor: Dr. D. David Hernandez Lopez.
david.hernandez@ulcm.es
Migración a Python: Juan Pedro García Tendero
juanpedrogarcia@terra.es

Versión: 1.0 para Python 2.7.

Última Modificación: Agosto 2011

egm08rednap. – fichero que almacena la clase Egm08rednap para el manejo de geoides en formato egm08rednap.

Define la función getOndulation(self,latitud,longitud).
Llama al fichero geoide que a su vez llama a los ficheros EGM08_REDNAP

```
entrada=(latitud,longitud)
salida=ondulación
import os,sys
from numpy import fromfile, reshape
from geoide import Geoide
```

geoide. – fichero que almacena la clase Geoide para el manejo de geoides en formato egm08rednap.

La clase Geoide almacena "llama" a los geoides EGM08_REDNAP de la Península o Baleares según corresponda

Define las funciones getNoOndulación y getOndulación en función de que encuentre las coordenadas de entrada dentro de los ficheros.

```
entrada=(latitud,longitud)
salida=ondulación
import os,sys
from numpy import fromfile, reshape,pi,floor,ndarray
```

ntv2. – fichero que almacena la función ntv2Espana.
Llama al fichero rejilla del IGN, en este caso sped2et, lo recorre y obtiene la longitud y latitud en deg en sistema ETRS89. Descargado de la web http://www.ikeralbeniz.net/2010/12/15/conversion-de-ed50-a-wgs84-o-etrs89/ (web de Iker Pérez de Albeniz)

```
entrada=(longitud,latitud) en deg en ED50
salida=(longitud, latitud) en deg en ETRS89
from numpy import pi
import math
from geotopo.general.rad2psdo import rad2psdo
```

ntv2_2. – fichero que almacena la función ntv2Espana. Adaptación del fichero anterior. Llama al fichero rejilla del IGN, en este caso sped2et, lo recorre y obtiene la longitud y latitud en deg en sistema ETRS89. Descargado de la web http://www.ikeralbeniz.net/2010/12/15/conversion-de-ed50-a-wgs84-o-etrs89/ (web de Iker Pérez de Albeniz)

En la adatación realizada, obtenemos sólo los diferenciales de cálculo de paso de ED50 a ETRS89 para tomarlo en el cálculo de la aproximación del sentido inverso entre ETRS89 a ED50. El paso de nodos del fichero utilizado es cada 200' lo que equivaldría a 6000 m aproximadamente por lo que la aproximación sería válida.

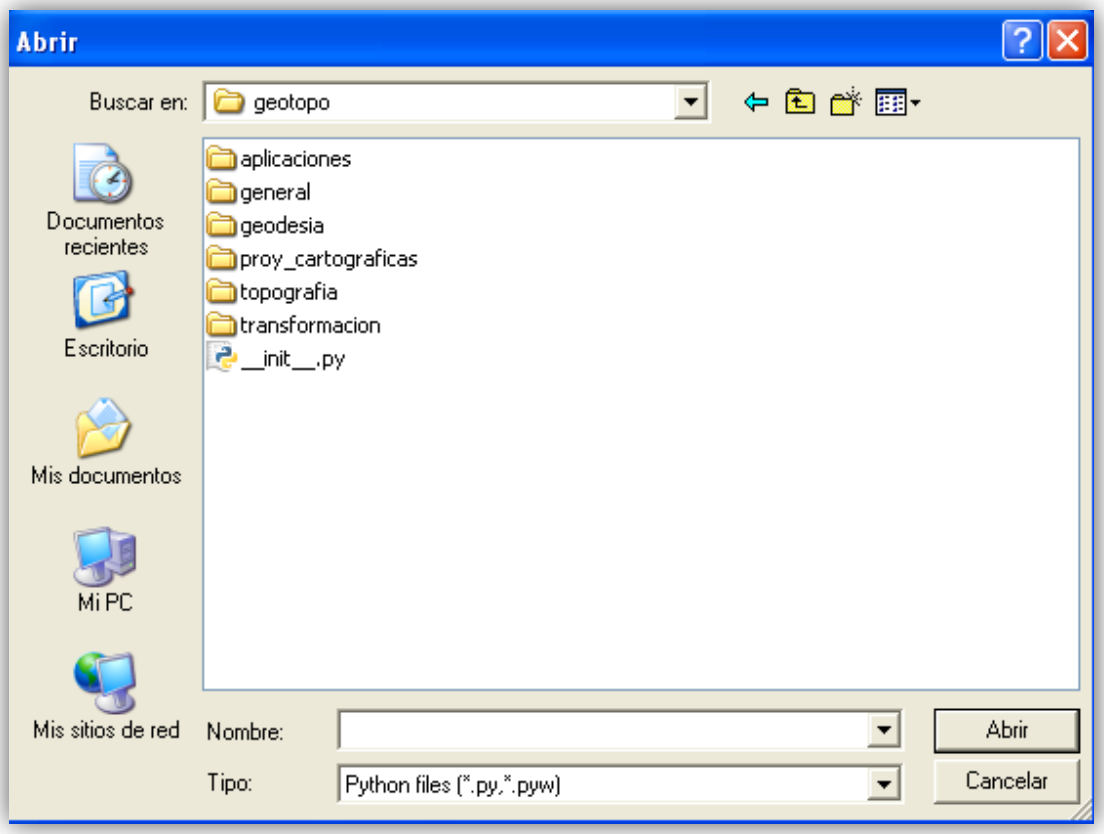
```
entrada=(longitud,latitud) en deg en ETRS89
salida=(diflongitud, diflatitud) en deg en ED50
from numpy import pi
import math
from geotopo.general.rad2psdo import rad2psdo
```

sped2et.py – fichero rejilla del IGN. Para transformación de coordenadas en los Sistemas de Referencia ED50 a ETRS89. Adaptado a Python y descargado de la web http://www.ikeralbeniz.net/2010/12/15/conversion-de-ed50-a-wgs84-o-etrs89/
(web de Iker Pérez de Albeniz)

entrada=(fichero contenedor de información). No entramos con ningún valor

5 APLICABILIDAD DE LA LIBRERÍA Y CONCLUSIONES FINALES

Finalmente, hemos implementado los 5 subpaquetes de módulos, con funciones y clases incluidos en el paquete (librería) geotopo destinados a cálculos topográficos y geodésicos.



Estructura final de la librería geotopo

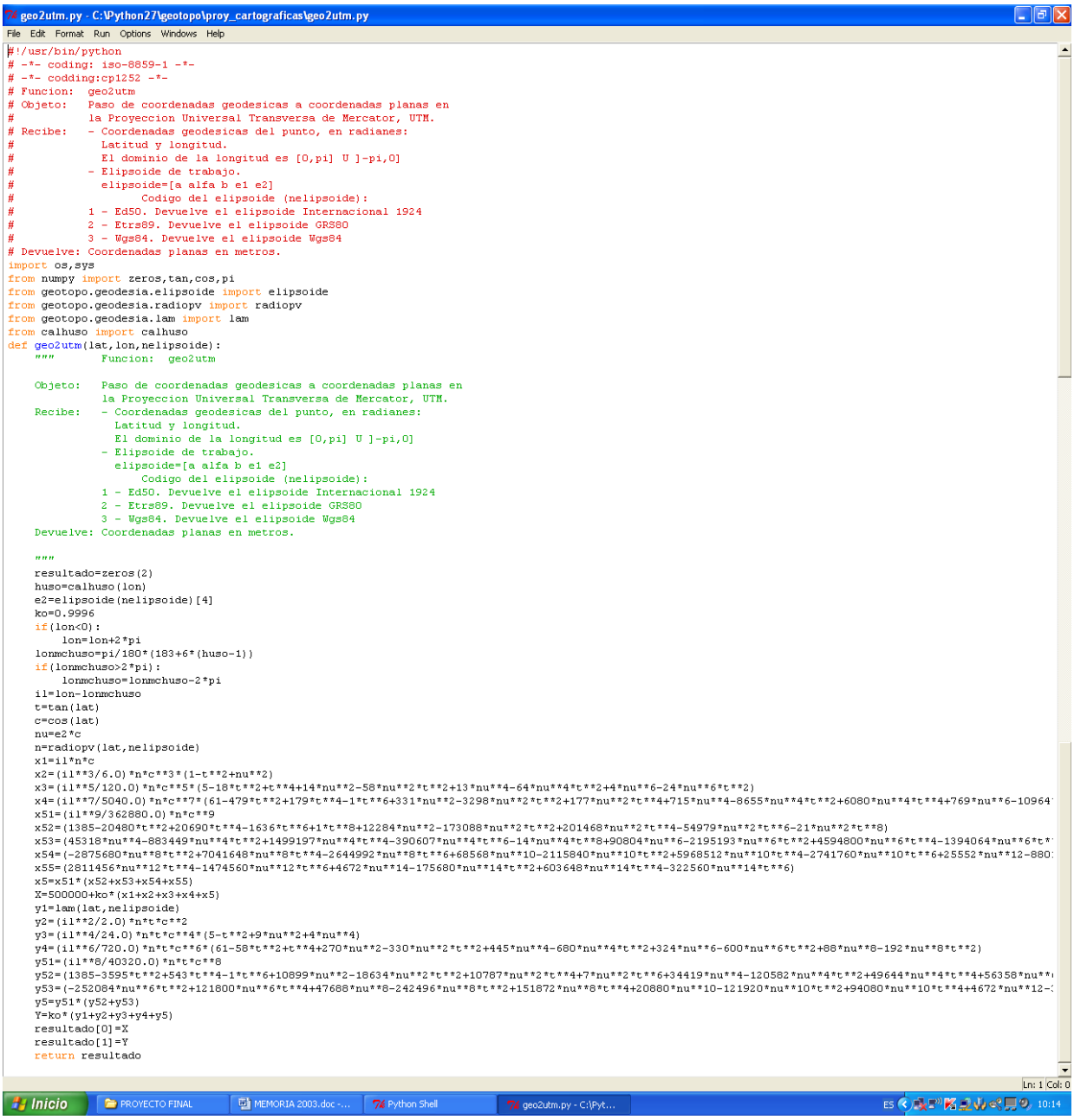
Como puede verse en el paquete de geotopo se ha incluido un módulo de aplicaciones en las que se han implementado unos ejercicios prácticos aplicando las distintas funciones de la librería. Antes de entrar a describir los casos prácticos implementados hemos de decir que directamente podemos obtener resultados de cada una de las funiones. Así si queremos pasar de coordenadas geográficas a coordenadas utm, mirando en la descripción de las funciones tenemos:

geo2utm. - paso de un punto, del elipsoide, a la proyección UTM, en el huso que le corresponde al punto por su longitud geodésica.

```
entrada=(lat,lon,nelipsoide)
salida=[X,Y] UTM
```

```
import os,sys
from numpy import zeros,tan,cos,pi
from calhuso import calhuso.
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.geodesia.radiopv import radiopv
from geotopo.geodesia.lam import lam
```

Pues bien, una vez seleccionada la función a utilizar, abrimos el módulo correspondiente con IDLE.



Código de la función abierto con IDLE

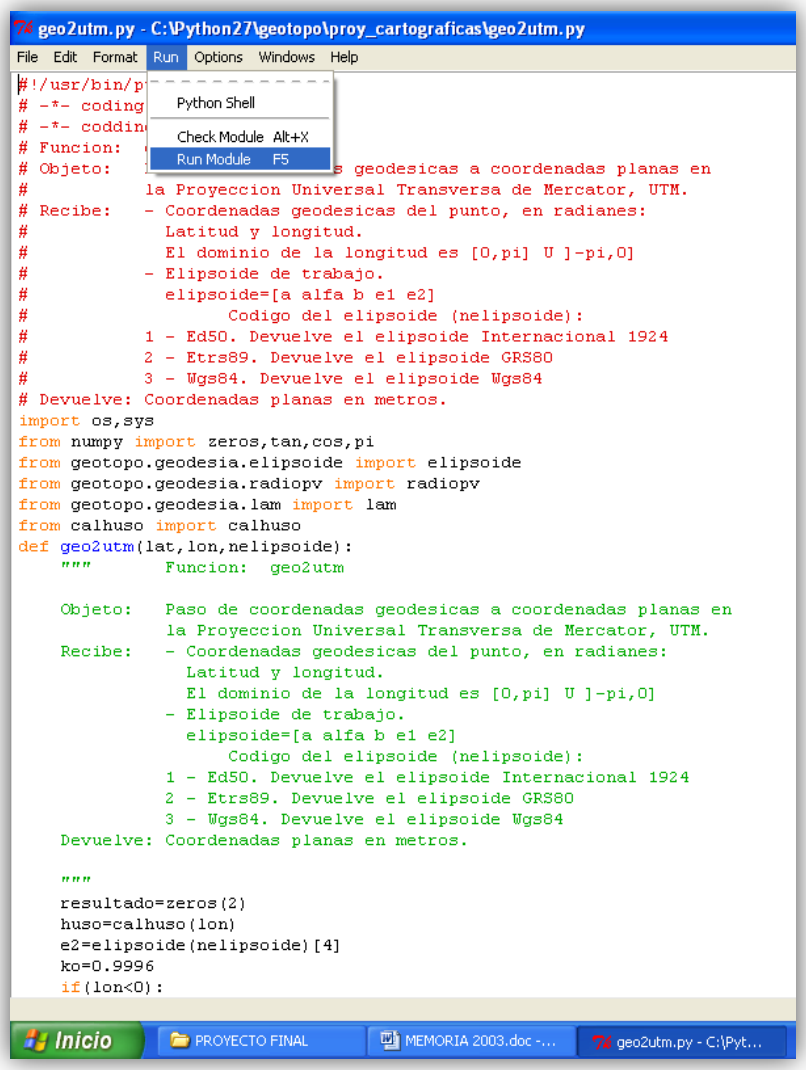
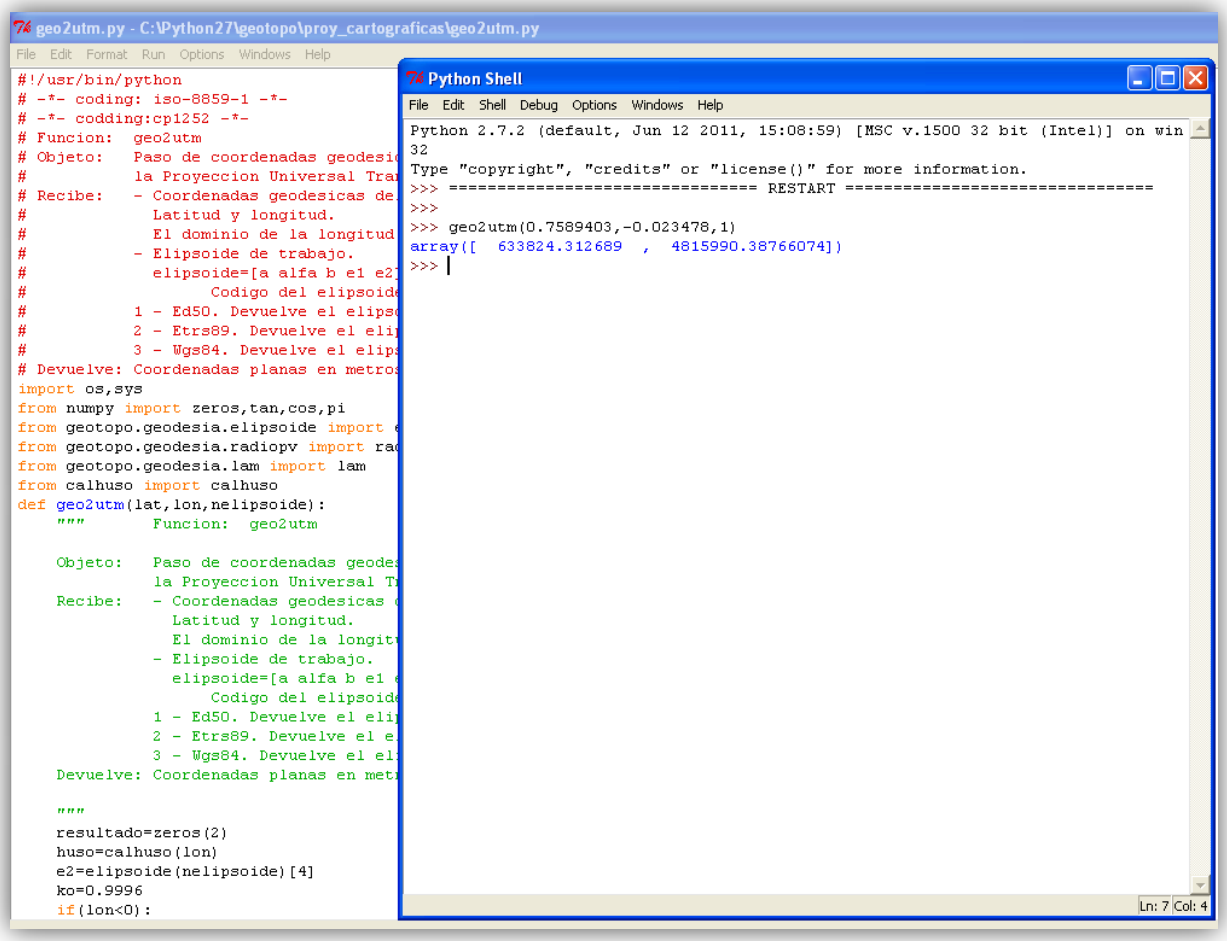


Imagen de la ejecución de Run Module

Ejecutando Run Module, Python abre el Python Shell. En esta pantalla, tenemos que introducir el nombre de la función a usar geo2utm y entre paréntesis los valores que se especifican de entrada en la descripción de la función entrada=(lat,lon,nelipsoide)

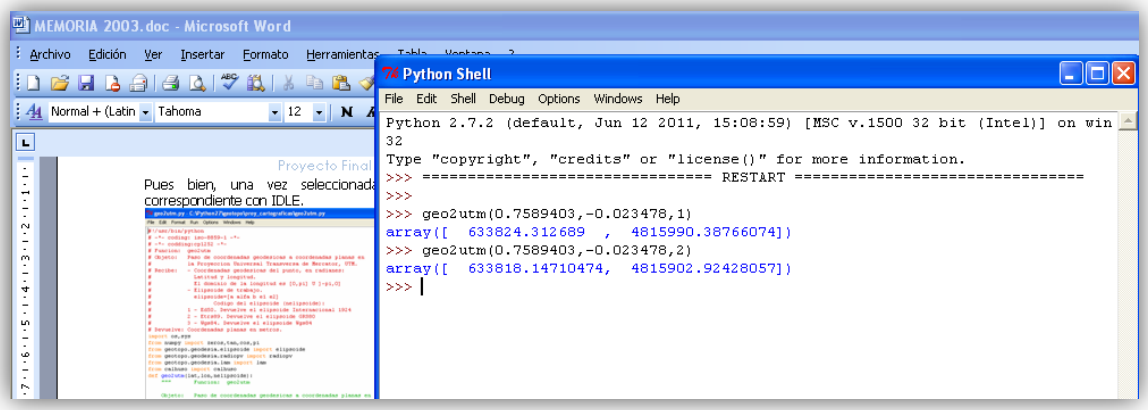


Llamada a la ejecución de la función desde Python Shell.

Como hemos indicado antes y por su relevancia y empleabilidad en otras funciones con "nelipsoide" se entiende un código del 1 al 3, que representa cada uno un elipsoides de referencia según los Sistemas de Referencia Coordenados, en este caso 1=ED50, elipsoide Internacional Hayford 1924, 2=ETRS89, elipsoide GRS80 y 3=WGS84.

Al introducir los datos, ejecutamos con Intro. Obtenemos 2 valores en forma de vector que respresentan según la descripción de la función salida=[X,Y] UTM.

Podemos volver a cálcular las coordenadas UTM en el SCR ETRS89, (en caso de que lalitud y longitud vinieran dadas en el mismo…) simplemente copiando la línea de entrada, pegándola abajo y cambiando 1 por 2 como índice del elipsoide y volviendo a ejecutar con enter.



Dentro de Python Shell, se puede copiar código de líneas anteriores y pegar abajo.

Para cálculos más complejos, de mayor número de puntos, etc, a modo de práctica se ha implementado el código de distintos tipos de cálculo de ejemplo. Distribuidos en 2 tipologías diferentes "topografia" y "geodesia" se han incluido códigos para la resolución, e impresión de distintos tipos de problemas realizando "import" de funciones declaradas en los módulos de los distintos subpaquetes de geotopo.

5.1 CASOS PRÁCTICOS TOPOGRAFÍA.

En total se han realizada 4 casos prácticos dentro del apartado de topografía. Todos ellos, tanto los ficheros .py, como los ficheros de datos de cálculos y salida de datos se incluyen en la versión digital de esta memoria. Sólo se muestra el código de la práctica topo_practica1. Del resto, sólo se ha incluido la salida de datos.

5.1.1 Implementación del código, topo_práctica1

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
# -*- coding: cp1252 -*-
# Práctica 1: CÁLCULOS TOPOGRÁFICOS ELEMENTALES
# Datos: Coordenadas del punto de estación
#        Sistema de referencia local
#'SEMANA GEOMÁTICA - 2011. Bogotá D.C. Colombia\n')
#'TALLER .......: Cálculos topográficos con software libre, librería GeoTop\n')
#'AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es\n')
#'                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es\n')
```

```
#
#'MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es\n'
#
# En esta versión de la práctica utilizando la librería geotopo en el cálculo de azimutes,
perímetro y superficie.

import os,sys
from numpy import matrix,array,shape,pi,zeros,concatenate,mean,sin,cos,sqrt
from geotopo.topografia.acimut import acimut
from geotopo.general.perimetro import perimetro
from geotopo.general.areacor import areacor

#Radio medio terrestre en metros.
rt=6370000.0

#Información del punto de estación. El significado de cada columna es:
#    - Primera columna.- Número entero identificador del punto de estación
#    - Segunda columna.- Coordenada X en metros.
#    - Tercera columna.- Coordenada Y en metros.
#    - Cuarta columna.- Altitud del punto, en metros.
#    - Quinta columna.- Altura de instrumento, en metros.
pe=array([[1000,3024.266,3090.376,657.208,1.50]])

#Información de las referencias:
# - Cada fila es un punto de referencia
# - El significado de cada columna es:
#    - Primera columna.- Número entero identificador del punto
#    - Segunda columna.- Coordenada X en metros.
#    - Tercera columna.- Coordenada Y en metros.
#    - Cuarta columna.- Lectura azimutal en grados centesimales.
ref=array([[2001,2786.397,3359.492,35.4821],[2002,3221.749,3275.750,133.5646],[2003,3289.406,282
6.990,231.3540],[2004,2612.221,2682.745,331.9067]])

#Información de los puntos radiados
# - Cada fila es una observación clásica a un punto radiado
# - El significado de cada columna es:
#    - Primera columna.- Número entero identificador del punto
#    - Segunda columna.- Lectura azimutal en grados centesimales.
#    - Tercera columna.- Lectura cenital en grados centesimales.
#    - Cuarta columna.- Distancia geométrica, en metros.
#    - Quinta columna.- Altura de instrumento, en metros.
rad=array([[3001,33.8713,100.1173,422.431,1.80],[3002,118.7710,100.0083,399.617,2.00],[3003,240.
5756,99.9039,440.097,1.50],[3004,330.3783,100.0468,414.996,1.50]])


#dimensiones [n° de filas, n° de columnas] de la matriz de puntos de referencia
# Cálculo de las dimensiones de la matriz de puntos de referencia:
# - mref será el número de filas, el número de puntos.
# - nref será el número de columnas, 4.
[mref,nref]=ref.shape
#Se extrae la información del punto de estación
num_pe=pe[0,0]
x_pe=pe[0,1]
y_pe=pe[0,2]
z_pe=pe[0,3]
alti_pe=pe[0,4]

#Cálculo de azitutes del punto de estación a cada uno de los puntos radiados
#La función azimut de la librería Geotop devuelve el valor en radianes
acimutes=zeros((mref,1))
for i in range (1,mref+1):
    #Se leen la identificación y las coordenadas del punto radiado y se almacenan en variables
temporales
    num_pref=ref[i-1,0]
    x_pref=ref[i-1,1]
    y_pref=ref[i-1,2]
    azi_pe_pref=acimut(x_pe,y_pe,x_pref,y_pref)
    acimutes[i-1,0]=azi_pe_pref*200.0/pi
ref=concatenate((ref,acimutes),axis=1)
```

```
#Cálculo de la desorientación de cada direcciónç
desor=zeros((mref,1))
for i in range(1,mref+1):
    #Se leen los valores angulares y se transforman a radianes, almacenándose en variables
temporales
    lhz_pe_pref=ref[i-1,3]*pi/200.0
    azi_pe_pref=ref[i-1,4]*pi/200.0
    des_pe_pref=azi_pe_pref-lhz_pe_pref
    if des_pe_pref<0:
        des_pe_pref=des_pe_pref+2.0*pi
    desor[i-1,0]=des_pe_pref*200/pi
ref=concatenate((ref,desor),axis=1)


#Cálculo de la desorientación media con la función de python
des_mean=mean(ref[:,5])*pi/200

#dimensiones [n° de filas, n° de columnas] de la matriz de puntos radiados
[mrad,nrad]=rad.shape

#Cálculo de coordenadas polares que se almacenan en la matriz coor_polares
#    - Primera columna.- Número entero identificador del punto radiado.
#    - Segunda columna.- Azimut en grados centesimales.
#    - Tercera columna.- Distancia reducida en metros.
#Cálculo de coordenadas cartesianas que se almacenan en la matriz coor_cart
#    - Primera columna.- Número entero identificador del punto del punto radiado.
#    - Segunda columna.- Coordenada X.
#    - Tercera columna.- Coordenada Y.
#    - Cuarta columna.- Altitud.
#antes se almacenan las coordenadas polares en la matriz coord_polares (numero de
punto,acimut,distancia reducida)
coord_polares=zeros((mrad,3))
coord_cart=zeros((mrad,4))
for i in range(1,mrad+1):
    num_prad=rad[i-1,0]
    lhz_pe_prad=rad[i-1,1]*pi/200.0
    lv_pe_prad=rad[i-1,2]*pi/200.0
    dg_pe_prad=rad[i-1,3]
    altp_pe_prad=rad[i-1,4]

    coord_polares[i-1,0]=num_prad
    azi_pe_prad=lhz_pe_prad+des_mean
    if azi_pe_prad>(2.0*pi):
        azi_pe_prad=azi_pe_prad-2.0*pi
    coord_polares[i-1,1]=azi_pe_prad*200.0/pi
    dr_pe_prad=dg_pe_prad*sin(lv_pe_prad)
    coord_polares[i-1,2]=dr_pe_prad

    coord_cart[i-1,0]=num_prad
    coord_cart[i-1,1]= x_pe+dr_pe_prad*sin(azi_pe_prad)
    coord_cart[i-1,2]= y_pe+dr_pe_prad*cos(azi_pe_prad)
    coord_cart[i-1,3]=                                  z_pe+alti_pe+dg_pe_prad*cos(lv_pe_prad)-
altp_pe_prad+0.42*dr_pe_prad**2/rt


#Cálculo del perímetro de la parcela utilizando la función perímetro de Geotopo
coord=zeros((mrad,2))
for i in range (1,mrad+1):
    coord[i-1,0]=coord_cart[i-1,1]
    coord[i-1,1]=coord_cart[i-1,2]
    perim=perimetro(coord);

#Cálculo de la superficie de la parcela utilizando la función areacor de Geotopo
superficie=areacor(coord)
#superficie_error=areacore(coord);

#Impresión de resultados
fsalida=open('topo.practica1.sal','w')
```

```
# Impresión de la cabecera del fichero de salida
fsalida.write('SEMANA GEOMÁTICA - 2011. Bogotá D.C. Colombia\n')
fsalida.write('TALLER .......: Cálculos topográficos con software libre, librería GeoTop\n')
fsalida.write('AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es\n')
fsalida.write('                              Dra. Beatriz Felipe García, bfelipe@jccm.es -
beatriz.felipe@uclm.es\n')
fsalida.write('MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es\n')
fsalida.write('\nPRÁCTICA 1 ...: CÁLCULOS TOPOGRÁFICOS ELEMENTALES\n')
fsalida.write('- Nota.- Todos los ángulos se expresan en graduación centesimal\n')
fsalida.write('- Nota.- Las magnitudes lineales se expresan en metros.\n')
fsalida.write('- Sistema de Referencia .................: Local\n')

#Impresión del paso 1: Paso de coordenadas cartesianas a polares: Azimutes
fsalida.write('\n1.  Cálculo  de  los  acimutes  a  la  referencia  y  desorientación  de  cada
visual.\n');
fsalida.write(' Punto Inicial     Punto Final      Acimut      Desorientación\n');
for i in range(1,mref+1):
    num_pref=ref[i-1,0]
    azi=ref[i-1,4]
    des=ref[i-1,5]
    fsalida.write('%10.0f'%num_pe)
    fsalida.write('%15.0f'%num_pref)
    fsalida.write('%18.4f'%azi)
    fsalida.write('%18.4f\n'%des)

#Impresión del paso 2: Cálculo de la desorientación media.
fsalida.write('\n2. Cálculo de la desorientación media.\n');
fsalida.write(' Desorientación media:%15.4f\n'%(des_mean*200/pi));

#Impresión del paso 3: Cálculo de la coordenadas polares.
fsalida.write('\n3. Cálculo de las coordenadas polares.\n');
fsalida.write('     Punto           Acimut           Distancia\n');
for i in range (1,mrad+1):
    num_prad=coord_polares[i-1,0]
    azi=coord_polares[i-1,1]
    dist=coord_polares[i-1,2]
    fsalida.write('%10.0f'%num_prad)
    fsalida.write('%18.4f'%azi)
    fsalida.write('%18.4f\n'%dist)

#Impresión del paso 4: Cálculo de la coordenadas cartesianas.
fsalida.write('\n4. Cálculo de las coordenadas cartesianas.\n')
fsalida.write('     Punto          X              Y              Z\n')
for i in range (1,mrad+1):
    num_prad=coord_cart[i-1,0]
    x=coord_cart[i-1,1]
    y=coord_cart[i-1,2]
    z=coord_cart[i-1,3]
    fsalida.write('%10.0f'%num_prad)
    fsalida.write('%15.4f'%x)
    fsalida.write('%15.4f'%y)
    fsalida.write('%15.4f\n'%z)

#Impresión del paso 5 valor del perímetro.
fsalida.write('\n5. Cálculo del perímetro de la parcela.\n')
fsalida.write(' Perímetro:%15.3f m.\n'%perim)

#Impresión del paso 6 valor de la superficie de la parcela.
fsalida.write('\n6. Cálculo de la superficie de la parcela.\n')
fsalida.write(' Superficie:%15.3f m2.\n'%superficie)
fsalida.close()
```

5.1.2 Impresión del fichero de salida de resultados, topo_práctica1

Ejecuando el programa anterior obtenemos un fichero con los resultados y la presentación mejorada que hemos implementado.

```
SEMANA GEOMÁTICA - 2011. Bogotá D.C. Colombia
TALLER .......: Cálculos topográficos con software libre, librería GeoTop
AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es

PRÁCTICA 1 ...: CÁLCULOS TOPOGRÁFICOS ELEMENTALES
- Nota.- Todos los ángulos se expresan en graduación centesimal
- Nota.- Las magnitudes lineales se expresan en metros.
- Sistema de Referencia .................: Local

1. Cálculo de los acimutes a la referencia y desorientación de cada visual.
 Punto Inicial     Punto Final      Acimut      Desorientación
      1000           2001          353.9187          318.4366
      1000           2002           52.0128          318.4482
      1000           2003          149.7887          318.4347
      1000           2004          250.3428          318.4361

2. Cálculo de la desorientación media.
 Desorientación media:       318.4389

3. Cálculo de las coordenadas polares.
     Punto           Acimut           Distancia
      3001          352.3102          422.4303
      3002           37.2099          399.6170
      3003          159.0145          440.0965
      3004          248.8172          414.9959

4. Cálculo de las coordenadas cartesianas.
     Punto          X              Y              Z
      3001      2736.5966      3399.7199      656.1414
      3002      3244.7650      3423.6535      656.6664
      3003      3288.4285      2738.3773      657.8851
      3004      2736.3219      2791.5286      656.9143

5. Cálculo del perímetro de la parcela.
 Perímetro:       2358.248 m.

6. Cálculo de la superficie de la parcela.
 Superficie:     342540.930 m2.
```

5.1.3 Salida de resultados topo_práctica2

```
SEMANA GEOMÁTICA - 2011. Bogotá D.C. Colombia
TALLER .......: Cálculos topográficos con software libre, librería GeoTop
AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN PYTHON:   Juan Pedro García Tendero. juanpedrogarcia@terra.es

PRÁCTICA 2 ...: MÉTODOS TOPOGRÁFICOS. CÁLCULO DE UNA INTERSECCIÓN DIRECTA
- Nota.- Todos los ángulos se expresan en graduación centesimal
- Nota.- Las magnitudes lineales se expresan en metros.
- Sistema de Referencia .................: Local

COORDENADAS DEL PUNTO DESCONOCIDO
X=  3264.596 m.
```

```
Y=  4065.437 m.

Previsión de error planimétrico=    0.035 m.

Vector con soluciones en z y su error:
Z=725.931 m    Previsión de error altimétrico     0.023 m.
Z=725.926 m    Previsión de error altimétrico     0.024 m.

SEMANA GEOMÁTICA - 2011. Bogotá D.C. Colombia
TALLER .......: Cálculos topográficos con software libre, librería GeoTop
AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN PYTHON:   Juan Pedro García Tendero. juanpedrogarcia@terra.es

PRÁCTICA 2 ...:MÉTODOS TOPOGRÁFICOS. CÁLCULO DE UNA INTERSECCIÓN INVERSA
- Nota.- Todos los ángulos se expresan en graduación centesimal
- Nota.- Las magnitudes lineales se expresan en metros.
- Sistema de Referencia ................: Local

COORDENADAS DEL PUNTO DESCONOCIDO
X=  3010.146 m.
Y=  3411.841 m.

Previsión de error planimétrico=    0.022 m.

Vector con soluciones en z y su error:
Z=515.695 m    Previsión de error altimétrico     0.021 m.
Z=515.703 m    Previsión de error altimétrico     0.024 m.
Z=515.698 m    Previsión de error altimétrico     0.022 m.
```

5.1.4 Salida de resultados topo_práctica3

```
SEMANA GEOMÁTICA - 2011. Bogotá D.C. Colombia
TALLER .......: Cálculos topográficos con software libre, librería GeoTop
AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es
PRÁCTICA 3 ...: CALCULO DE DATOS DE REPLANTEO PARA ESTACION TOTAL EN UN SISTEMA DE REFERENCIA
LOCAL
- Nota.- Todos los ángulos se expresan en graduación centesimal
- Nota.- Las magnitudes lineales se expresan en metros.
- Sistema de Referencia ................: Local

Resultados:Datos de replanteo
Base    Punto       Azimut    Distancia   CotaProyecto    CotaTerreno   CotaRoja
   1        1    385.6489      75.460      96.930          96.539      0.391
   1        2    266.7262      77.786      96.903          98.259     -1.356
   1        3    121.9144      40.883      95.329          94.688      0.641
   1        4     27.9600      67.437      95.045          95.587     -0.542
```

5.1.5 Salida de resultados topo_práctica4

```
SEMANA GEOMÁTICA - 2011. Bogotá D.C. Colombia
TALLER .......: Cálculos topográficos con software libre, librería GeoTop
AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es
PRÁCTICA 4 ...: TRANSFORMACIÓN ENTRE SISTEMAS DE REFERENCIA LOCALES: HELMERT 2D
- Nota.- Todos los ángulos se expresan en graduación centesimal
- Nota.- Las magnitudes lineales se expresan en metros.
- Sistema de Referencia ................: Local

Planteamiento del sistema de ecuaciones que define la transformación planimétrica:
 Matriz A
         1.000         0.000      3258.686     -2709.414
         0.000         1.000      2709.414      3258.686
         1.000         0.000      2075.590     -2105.320
         0.000         1.000      2105.320      2075.590
 Vector términos independientes
      4554.780
       228.501
      3244.004
       443.737

1. Resultados de los parámetros de la transformación:
- Planimetría:
  - Traslación en coordenada x del sistema S1 al sistema S2 ....................:      324.401 m.
  - Traslación en coordenada y del sistema S1 al sistema S2 ....................:      -20.424 m.
  - Valor para el parámetro a (a=landa*cos_giro) ..............................:   0.805119725
adimensional.
  - Valor para el parámetro b (b=landa*sin_giro) ..............................: -0.593023724
adimensional.
  - Valor para el factor de escala (landa=sqrt(a**2+b**2)) .....................:      0.999947
tanto por uno.
  - Valor para el giro (giro=arctan2(b,a) .....................................:      -40.4157
grados centesimales.
- Altimetría:
  - Traslación en coordenada z del sistema S1 al sistema S2 para el punto 1 ...:        6.056 m.
  - Traslación en coordenada z del sistema S1 al sistema S2 para el punto 2 ...:        6.083 m.
  - Traslación media en coordenada z del sistema S1 al sistema S2 .............:        6.070 m.

2. Resultado de la transformación del punto 3 al sistema de referencia local S2:
- Coordenada x de 3 en el sistema de referencia local S2 .....:     4072.286 m.
- Coordenada y de 3 en el sistema de referencia local S2 .....:       45.170 m.
- Coordenada z de 3 en el sistema de referencia local S2 .....:      104.748 m.
```

5.2 CASOS PRÁCTICOS GEODESIA.

En total se han realizada 5 casos prácticos dentro del apartado de geodesia. Todos ellos, tanto los ficheros .py, como los ficheros de datos de cálculos y salida de datos se incluyen en la versión digital de esta memoria. Sólo se muestra el código de la práctica geo_practica1. Del resto, sólo se ha incluido la salida de datos.

5.2.1 Implementación del código, geo_práctica1

```
# Semana Geomática 2011, Bogotá D.C., Colombia
# IGAC-Instituto Agustín Codazzi
# Taller: Cálculos geodésicos y de cartografía matemática con software libre, librería GeoTop
# Impartido por: Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
#                Dr. David Hernández López, david.hernandez@ulcm.es
#
#'MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es\n'
#
#  Práctica 1: CÁLCULOS GEODÉSICOS ELEMENTALES - OPERACIONES DE CONVERSIÓN ENTRE COORDENADAS GEODÉSICAS Y ECEF
#  Datos: Coordenadas de la Red ERGNSS - Red de Estaciones Permanentes GNSS - Instituto Geográfico Nacional, España
#         Sistema de referencia: ETRS89 (Elipsoide GRS80)
#         Código EPSG para coordenadas geodésicas sobre el elipsoide: 4258
# En esta versión de la práctica se utiliza la librería GeoTop.
# Otra variante con respecto a la versión anterior de la práctica es que se parte de coordenadas geodésicas en seudo decimal sexagesimal.
#  Documentación: La formulación empleada figura en el apartado 1.3 "Parametrización del elipsoide" del documento:
#                \Octave\TallerGeodesia\Documentacion\Apuntes_DHL_2010\GeodesiaElipsoidal.pdf
#
# Matriz de coordenadas:
# - Cada fila es un punto, una estación de referencia GNSS.
# - El significado de cada columna es:
#   - Primera columna.- Número entero identificador del punto
#   - Segunda columna.- Coordenada geodésica longitud, en suedo decimal sexagesimal.
#   - Tercera columna.- Coordenada geodésica latitud, en suedo decimal sexagesimal.
#   - Cuarta columna.- Altitud elipsoidal, en metros.
from numpy import zeros,array,genfromtxt
from geotopo.geodesia.elipsoide import elipsoide
from geotopo.general.rad2psdo import rad2psdo
from geotopo.general.psdo2rad import psdo2rad
from geotopo.geodesia.tri2geo import tri2geo
from geotopo.geodesia.geo2tri import geo2tri
#c4258h=[       1   43.2151770811   -8.2356167472      66.917
#       2   38.2020103584   -0.2852437048      60.356
#       3   38.5840494457   -1.5123054192     751.736
#       4   36.5109112636   -2.2734018800     127.517
#       5   39.2843898433   -6.2030426066     436.527
#       6   43.2819118007   -3.4753038539      99.287
#       7   35.5331090227   -5.1823016758      52.475
#       8   37.5456176191   -4.4315999947     202.082
#       9   37.1159923466   -6.5513077151      81.860
#      10   40.2645009016   -3.4234283233     766.920
#      11   42.3518259367   -5.3903511817     970.254
#      12   36.4334003748   -4.2336718238     119.848
#      13   39.3309445258    2.3728383134      62.063
#      14   42.2728617761   -2.3007691629     503.204
#      15   40.5642292231   -5.2945108842     854.969
#      16   39.4031263811   -3.5750290360     808.968
#      17   40.2101790155   -1.0727483699     956.188
#      18   39.2850970441   -0.2015543423      77.592
#      19   42.1102318373   -8.4847057525      87.790
#      20   40.3129631681   -3.0519065001     972.777
#      21   41.3800221985   -0.5255792567     296.111
#      22   28.1829034024  -16.2958855078    2417.483
#      23   28.2837857413  -16.1428164273      51.787
#      24   28.2505711067  -16.3302841359      54.475
#      25   28.0249784309  -16.4306688806      58.543
#      26   28.4549932439  -17.5337787438    2199.221];
# Los datos, los guardamos en un txt en la misma carpeta del trabajo e importamos
c4258h=genfromtxt("C:\\Python27\\geotopo\\aplicaciones\\talleres\geo_practica1dat.txt")
# Cálculo de las dimensiones de la matriz de coordenadas:
# - m será el número de filas, el número de puntos.
# - n será el número de columnas, 4.
```

```
[m,n]=c4258h.shape

grs80=elipsoide(2)
# Recibe:   Codigo del elipsoide (nelipsoide):
#           1 - Ed50. Devuelve el elipsoide Internacional 1924
#           2 - Etrs89. Devuelve el elipsoide GRS80
#           3 - Wgs84. Devuelve el elipsoide Wgs84
# Definición del elipsoide
fsalida=open('geo_practica1.sal','w')
fsalida.write('SEMANA GEOMÁTICA - 2011\n')
fsalida.write('TALLER .......: Cálculos geodésicos y de cartografía matemática con software
libre, librería GeoTop\n')
fsalida.write('AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es\n')
fsalida.write('                                Dra. Beatriz Felipe García, bfelipe@jccm.es -
beatriz.felipe@uclm.es\n')
fsalida.write('MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es\n')
fsalida.write('PRÁCTICA 1 ...: CÁLCULOS GEODÉSICOS ELEMENTALES - OPERACIONES DE CONVERSIÓN ENTRE
COORDENADAS GEODÉSICAS Y ECEF\n')
fsalida.write('- Nota.- Todos los ángulos se expresan en seudo decimal sexagesimal:
gg.mmssfs\n')
fsalida.write('- Nota.- Las magnitudes lineales se expresan en metros.\n')
fsalida.write('- Sistema de Referencia .................: ETRS89\n')
fsalida.write('- Parámetros del elipsoide ..............: GRS80\n')
fsalida.write('  - Semieje mayor .......................:  %10.3f m.\n'%grs80[0])
fsalida.write('  - Inverso del aplanamiento ............:  %10.9f adim.\n'%grs80[1])
fsalida.write('  - Semieje menor .......................:  %10.3f m.\n'%grs80[2])
fsalida.write('  - Primera excentricidad al cuadrado ..:  %10.9f adim.\n'%grs80[3]**2)
fsalida.write('  - Segunda excentricidad al cuadrado ..:  %10.9f adim.\n'%grs80[4]**2)

# Impresión de la cabecera del fichero de salida
fsalida.write('1. Paso de coordenadas geodésicas a ECEF.\n')
fsalida.write('N.Punto          Latitud       Longitud    Alt.Elip.     Coor.X.ECEF     Coor.Y.ECEF
Coor.Z.ECEF\n')
c4258XYZ=zeros((m,4))
for i in range (1,m+1):
    np=c4258h[i-1,0]
    longitudRad=psdo2rad(c4258h[i-1,2])
    latitudRad=psdo2rad(c4258h[i-1,1])
    hElip=c4258h[i-1,3]
    [xecef,yecef,zecef]=geo2tri(latitudRad,longitudRad,hElip,2)
    fsalida.write('%8.0f'%np)
    fsalida.write('%15.9f'%c4258h[i-1,1])
    fsalida.write('%15.9f'%c4258h[i-1,2])
    fsalida.write('%12.3f'%hElip)
    fsalida.write('%15.3f'%xecef)
    fsalida.write('%15.3f'%yecef)
    fsalida.write('%15.3f'%zecef)
    fsalida.write('\n')
    c4258XYZ[i-1,0]=np
    c4258XYZ[i-1,1]=xecef
    c4258XYZ[i-1,2]=yecef
    c4258XYZ[i-1,3]=zecef

# Cálculo e impresión del paso 2: de coordenadas ECEF a geodésicas
fsalida.write('2. Paso de coordenadas ECEF a geodésicas.\n')
fsalida.write(' N.Punto      Coor.X.CG      Coor.Y.CG      Coor.Z.CG      Latitud       Longitud
Alt.Elip.\n')
for i in range(1,m+1):
    np=c4258XYZ[i-1,0]
    xecef=c4258XYZ[i-1,1]
    yecef=c4258XYZ[i-1,2]
    zecef=c4258XYZ[i-1,3]
    [latitud,longitud,hElip]=tri2geo(xecef,yecef,zecef,2)
    fsalida.write('%8.0f'%np)
    fsalida.write('%15.3f'%xecef)
    fsalida.write('%15.3f'%yecef)
    fsalida.write('%15.3f'%zecef)
    fsalida.write('%15.9f'%rad2psdo(latitud))
    fsalida.write('%15.9f'%rad2psdo(longitud))
```

```
    fsalida.write('%15.3f\n'%hElip)
fsalida.close()
```

5.2.2 Impresión del fichero de salida de resultados, geo_práctica1

```
SEMANA GEOMÁTICA - 2011
TALLER .......: Cálculos geodésicos y de cartografía matemática con software libre, librería
GeoTop
AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es
PRÁCTICA 1 ...: CÁLCULOS GEODÉSICOS ELEMENTALES - OPERACIONES DE CONVERSIÓN ENTRE COORDENADAS
GEODÉSICAS Y ECEF
- Nota.- Todos los ángulos se expresan en seudo decimal sexagesimal: gg.mmssfs
- Nota.- Las magnitudes lineales se expresan en metros.
- Sistema de Referencia ................: ETRS89
- Parámetros del elipsoide ..............: GRS80
  - Semieje mayor .....................:  6378137.000 m.
  - Inverso del aplanamiento ...........:  0.003352811 adim.
  - Semieje menor .....................:  6356752.314 m.
  - Primera excentricidad al cuadrado ..:  0.006694380 adim.
  - Segunda excentricidad al cuadrado ..:  0.006739497 adim.
1. Paso de coordenadas geodésicas a ECEF.
N.Punto         Latitud       Longitud   Alt.Elip.    Coor.X.ECEF    Coor.Y.ECEF    Coor.Z.ECEF
      1   43.215177081   -8.235616747      66.917    4594489.890    -678368.010    4357065.904
      2   38.202010358   -0.285243705      60.356    5009051.399     -42072.472    3935057.504
      3   38.584049446   -1.512305419     751.736    4962848.206    -160854.365    3990884.209
      4   36.510911264   -2.273401880     127.517    5105220.295    -219278.803    3804386.889
      5   39.284389843   -6.203042607     436.527    4899866.820    -544567.584    4033769.790
      6   43.281911801   -3.475303854      99.287    4625924.690    -307096.765    4365771.175
      7   35.533109023   -5.182301676      52.475    5150908.012    -478415.023    3718518.240
      8   37.545617619   -4.431599995     202.082    5021256.063    -414685.080    3898182.184
      9   37.115992347   -6.551307715      81.860    5049613.168    -612885.631    3835143.860
     10   40.264500902   -3.423428323     766.920    4851137.670    -314518.688    4116282.036
     11   42.351825937   -5.390351182     970.254    4680871.385    -463168.384    4294606.572
     12   36.433400375   -4.233671824     119.848    5103282.414    -392096.752    3793146.894
     13   39.330944526    2.372838313      62.063    4919369.704     225499.577    4039849.606
     14   42.272861776   -2.300769163     503.204    4708688.612    -205761.707    4283609.369
     15   40.564229223   -5.294510884     854.969    4803054.799    -462131.609    4158378.661
     16   39.403126381   -3.575029036     808.968    4904660.519    -339868.032    4050823.512
     17   40.210179016   -1.072748370     956.188    4867391.684     -95523.894    4108341.277
     18   39.285097044   -0.201554342      77.592    4929534.046     -29050.676    4033709.925
     19   42.110231837   -8.484705753      87.790    4677481.077    -725205.068    4260827.192
     20   40.312963168   -3.051906500     972.777    4848724.914    -261632.472    4123093.922
     21   41.380022199   -0.525579257     296.111    4773803.543     -73506.519    4215453.698
     22   28.182903402  -16.295885508    2417.483    5390243.531   -1596630.330    3007752.593
     23   28.283785741  -16.142816427      51.787    5386836.438   -1569217.617    3023118.925
     24   28.250571107  -16.330284136      54.475    5381262.061   -1599192.518    3017377.791
     25   28.024978431  -16.430668881      58.543    5395193.089   -1620537.044    2981146.536
     26   28.454993244  -17.533778744    2199.221    5326646.317   -1719826.438    3052043.561
2. Paso de coordenadas ECEF a geodésicas.
 N.Punto   Coor.X.CG      Coor.Y.CG      Coor.Z.CG      Latitud       Longitud         Alt.Elip.
       1  4594489.890    -678368.010    4357065.904   43.215177081   -8.235616747        66.917
       2  5009051.399     -42072.472    3935057.504   38.202010358   -0.285243705        60.356
       3  4962848.206    -160854.365    3990884.209   38.584049446   -1.512305419       751.736
       4  5105220.295    -219278.803    3804386.889   36.510911264   -2.273401880       127.517
       5  4899866.820    -544567.584    4033769.790   39.284389843   -6.203042607       436.527
       6  4625924.690    -307096.765    4365771.175   43.281911801   -3.475303854        99.287
       7  5150908.012    -478415.023    3718518.240   35.533109023   -5.182301676        52.475
       8  5021256.063    -414685.080    3898182.184   37.545617619   -4.431599995       202.082
       9  5049613.168    -612885.631    3835143.860   37.115992347   -6.551307715        81.860
      10  4851137.670    -314518.688    4116282.036   40.264500902   -3.423428323       766.920
      11  4680871.385    -463168.384    4294606.572   42.351825937   -5.390351182       970.254
```

12	5103282.414	-392096.752	3793146.894	36.433400375	-4.233671824	119.848
13	4919369.704	225499.577	4039849.606	39.330944526	2.372838313	62.063
14	4708688.612	-205761.707	4283609.369	42.272861776	-2.300769163	503.204
15	4803054.799	-462131.609	4158378.661	40.564229223	-5.294510884	854.969
16	4904660.519	-339868.032	4050823.512	39.403126381	-3.575029036	808.968
17	4867391.684	-95523.894	4108341.277	40.210179016	-1.072748370	956.188
18	4929534.046	-29050.676	4033709.925	39.285097044	-0.201554342	77.592
19	4677481.077	-725205.068	4260827.192	42.110231837	-8.484705753	87.790
20	4848724.914	-261632.472	4123093.922	40.312963168	-3.051906500	972.777
21	4773803.543	-73506.519	4215453.698	41.380022199	-0.525579257	296.111
22	5390243.531	-1596630.330	3007752.593	28.182903402	-16.295885508	2417.483
23	5386836.438	-1569217.617	3023118.925	28.283785741	-16.142816427	51.787
24	5381262.061	-1599192.518	3017377.791	28.250571107	-16.330284136	54.475
25	5395193.089	-1620537.044	2981146.536	28.024978431	-16.430668881	58.543
26	5326646.317	-1719826.438	3052043.561	28.454993244	-17.533778744	2199.221

5.2.3 Salida de resultados geo_práctica2

```
TALLER .......: Cálculos geodésicos y de cartografía matemática con software libre, librería GeoTop
AUTORES ......: Dr. David Heradiopvández López, david.heradiopvandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es
PRÁCTICA 2 ...: CÁLCULOS GEODÉSICOS SOBRE EL ELIPSOIDE
- Nota.- Las coordenadas geodésicas se expresan en seudo decimal sexagesimal: gg.mmssfs
- Nota.- Los azimutes se expresan en centesimal.
- Nota.- Las magnitudes lineales se expresan en metros.
- Nota.- Los errores y las correcciones lineales se expresan en milímetros.
- Sistema de Referencia ..............: ETRS89
- Parámetros del elipsoide ..........: GRS80
  - Semieje mayor ....................:  6378137.000 m.
  - Inverso del aplanamiento .........:  0.003352811 adim.
  - Semieje menor ....................:  6356752.314 m.
  - Primera excentricidad al cuadrado ..:  0.006694380 adim.
  - Segunda excentricidad al cuadrado ..:  0.006739497 adim.

Problema directo e inverso de la geodesia sobre el elipsoide.
  N.PI  N.PF     Latitud.PI     Longitud.PI     Latitud.PF     Longitud.PF      DisLinGeo   AziDirLinGeo
     1     4   43.215177081    -8.235616747   36.510911264    -2.273401880    882202.8933  158.932145196
     1     6   43.215177081    -8.235616747   43.281911801    -3.475303854    372749.3356   96.202059188
     1     9   43.215177081    -8.235616747   37.115992347    -6.551307715    695915.9527  187.896123886
     1    13   43.215177081    -8.235616747   39.330944526     2.372838313   1012559.6629  123.285930915
     1    16   43.215177081    -8.235616747   39.403126381    -3.575029036    552014.9003  151.568183719
     4     6   36.510911264    -2.273401880   43.281911801    -3.475303854    743773.1934  390.675618424
     4     9   36.510911264    -2.273401880   37.115992347    -6.551307715    398761.7472  307.654701848
     4    13   36.510911264    -2.273401880   39.330944526     2.372838313    536630.4339   60.560584194
     4    16   36.510911264    -2.273401880   39.403126381    -3.575029036    339863.1907  375.188557967
     6     9   43.281911801    -3.475303854   37.115992347    -6.551307715    745118.7945  224.311221621
     6    13   43.281911801    -3.475303854   39.330944526     2.372838313    690251.0547  141.007508821
     6    16   43.281911801    -3.475303854   39.403126381    -3.575029036    421899.2029  202.149794087
     9    13   37.115992347    -6.551307715   39.330944526     2.372838313    873458.5186   77.428010916
     9    16   37.115992347    -6.551307715   39.403126381    -3.575029036    376946.4598   46.997544544
    13    16   39.330944526     2.372838313   39.403126381    -3.575029036    565811.3244  303.866621083
```

AziDirLinGeo	AziInvLinGeo	Latitud.PF.Pd	Longitud.PF.Pd	Error.PF	Latitud.PI.Pd	Longitud.PI.Pd	Error.PI
158.932145196	363.192865269	36.510911264	-2.273401880	0.0007	43.215177081	-8.235616747	0.0007
96.202059188	299.716683037	43.281911801	-3.475303854	0.0020	43.215177081	-8.235616747	0.0020
187.896123886	388.959941135	37.115992347	-6.551307715	0.0022	43.215177081	-8.235616747	0.0021
123.285930915	331.413881708	39.330944526	2.372838313	0.0047	43.215177081	-8.235616747	0.0047
151.568183719	354.837297679	39.403126381	-3.575029036	0.0015	43.215177081	-8.235616747	0.0015
390.675618424	189.714711953	43.281911801	-3.475303854	0.0022	36.510911264	-2.273401880	0.0024
307.654701848	104.669013287	37.115992347	-6.551307715	0.0019	36.510911264	-2.273401880	0.0019
60.560584194	264.056499034	39.330944526	2.372838313	0.0016	36.510911264	-2.273401880	0.0016
375.188557967	174.152954299	39.403126381	-3.575029036	0.0024	36.510911264	-2.273401880	0.0025
224.311221621	22.062051066	37.115992347	-6.551307715	0.0006	43.281911801	-3.475303854	0.0006
141.007508821	345.742835256	39.330944526	2.372838313	0.0022	43.281911801	-3.475303854	0.0022
202.149794087	2.027403922	39.403126381	-3.575029036	0.0010	43.281911801	-3.475303854	0.0009
77.428010916	284.022885321	39.330944526	2.372838313	0.0037	37.115992347	-6.551307715	0.0037
46.997544544	249.040346243	39.403126381	-3.575029036	0.0003	37.115992347	-6.551307715	0.0003
303.866621083	99.195863187	39.403126381	-3.575029036	0.0025	39.330944526	2.372838313	0.0025

5.2.4 Salida de resultados geo_práctica3

```
SEMANA GEOMÁTICA - 2011
TALLER .......: Cálculos geodésicos y de cartografía matemática con software libre, librería GeoTop
AUTORES ......: Dr. David Heradiopvández López, david.heradiopvandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es
PRÁCTICA 3 ...: CONVERSIÓN ENTRE COORDENADAS GEODÉSICAS Y UTM
- Nota.- Las coordenadas geodésicas se expresan en seudo decimal sexagesimal: gg.mmssfs
- Nota.- Los valores angulares en la proyección UTM se expresan en centesimal.
- Nota.- Las magnitudes lineales se expresan en metros,
         salvo los errores que se expresan en milímetros.
- Sistema de Referencia ..............: ETRS89
- Parámetros del elipsoide ............: GRS80
  - Semieje mayor ....................:  6378137.000 m.
  - Inverso del aplanamiento .........:  0.003352811 adim.
  - Semieje menor ....................:  6356752.314 m.
  - Primera excentricidad al cuadrado ..:  0.006694380 adim.
  - Segunda excentricidad al cuadrado ..:  0.006739497 adim.

1. Conversión entre coordenadas geodésicas y UTM, con cada punto en su huso.
 N.Pto       Latitud       Longitud      Coor.X-UTM      Coor.Y-UTM  Huso     Kp-UTM        Conv.UTM  Latitud_DeUTM  Longitud_DeUTM    Error
     1   43.215177081   -8.235616747    548701.3416   4801455.3611    29  0.99962918    0.458578318   43.215177081   -8.235616747   0.0000
     4   36.510911264   -2.273401880    548188.9688   4078649.7274    30  0.99962861    0.360228168   36.510911264   -2.273401880   0.0009
     6   43.281911801   -3.475303854    435450.9229   4813538.5334    30  0.99965125   -0.610098073   43.281911801   -3.475303854   0.0000
     9   37.115992347   -6.551307715    684568.8666   4119082.7742    29  1.00001965    1.397488431   37.115992347   -6.551307715   0.0010
    13   39.330944526    2.372838313    467743.5663   4378172.0434    31  0.99961281   -0.265647838   39.330944526    2.372838313   0.0000
    16   39.403126381   -3.575029036    417326.7694   4392169.4070    30  0.99968415   -0.683853784   39.403126381   -3.575029036   0.0000

2. Conversión entre coordenadas geodésicas y UTM, con todos los puntos en el huso 30.
 N.Pto       Latitud       Longitud      Coor.X-UTM      Coor.Y-UTM  Huso     Kp-UTM        Conv.UTM  Latitud_DeUTM  Longitud_DeUTM    Error
     1   43.215177081   -8.235616747     62513.6627   4815454.5995    30  1.00195550   -4.125520664   43.215177081   -8.235616747   0.0608
     4   36.510911264   -2.273401880    548188.9688   4078649.7274    30  0.99962861    0.360228168   36.510911264   -2.273401880   0.0009
     6   43.281911801   -3.475303854    435450.9229   4813538.5334    30  0.99965125   -0.610098073   43.281911801   -3.475303854   0.0000
     9   37.115992347   -6.551307715    152029.2127   4124260.8849    30  1.00109200   -2.636207981   37.115992347   -6.551307715   0.0047
    13   39.330944526    2.372838313    983376.0812   4393239.8121    30  1.00247877    3.987301410   39.330944526    2.372838313   0.0641
    16   39.403126381   -3.575029036    417326.7694   4392169.4070    30  0.99968415   -0.683853784   39.403126381   -3.575029036   0.0000
```

5.2.5 Salida de resultados geo_práctica4

```
SEMANA GEOMÁTICA - 2011
TALLER .......: Cálculos geodésicos y de cartografía matemática con software libre, librería GeoTop
AUTORES ......: Dr. David Heradiopvández López, david.heradiopvandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es
PRÁCTICA 4 ...: PROYECCIÓN DE LÍNEAS GEODÉSICAS DEL ELIPSOIDE AL PLANO UTM
- Nota.- Las coordenadas geodésicas se expresan en seudo decimal sexagesimal: gg.mmssfs
- Nota.- Los valores angulares en la proyección UTM se expresan en centesimal,
         salvo los errores y correcciones que se expresan en segundos centesimales.
- Nota.- Las magnitudes lineales se expresan en metros,
         salvo los errores que se expresan en milímetros.
- Sistema de Referencia ................: ETRS89
- Parámetros del elipsoide .............: GRS80
  - Semieje mayor ......................:  6378137.000 m.
  - Inverso del aplanamiento ...........:  0.003352811 adim.
  - Semieje menor ......................:  6356752.314 m.
  - Primera excentricidad al cuadrado ..:  0.006694380 adim.
  - Segunda excentricidad al cuadrado ..:  0.006739497 adim.

1. Problema directo e inverso de la geodesia sobre el elipsoide.
  N.PI  N.PF     Latitud.PI    Longitud.PI     Latitud.PF    Longitud.PF      DisLinGeo   AziDirLinGeo   AziInvLinGeo  Latitud.PF.Pd Longitud.PF.Pd
 57070 59443   40.205473471    0.031681980   40.130390380    0.160281533     23201.6286  142.979350641  343.132210780   40.130390380    0.160281533
 57070 59341   40.205473471    0.031681980   40.114351399   -0.014806161     18464.9979  225.542238015   25.481410658   40.114351399   -0.014806161
 57070 64159   40.205473471    0.031681980   39.595563014   -0.000408439     39125.4972  207.773623558    7.733621949   39.595563014   -0.000408439
 59443 59341   40.130390380    0.160281533   40.114351399   -0.014806161     25446.1216  293.893422228   93.680057818   40.114351399   -0.014806161
 59443 64159   40.130390380    0.160281533   39.595563014   -0.000408439     33399.0292  248.190043635   47.997787001   39.595563014   -0.000408439
 59341 64159   40.114351399   -0.014806161   39.595563014   -0.000408439     21972.0673  192.838625464  392.859295298   39.595563014   -0.000408439

                                                  Longitud.PF.Pd   Error.PF  Latitud.PI.Pd Longitud.PI.Pd   Error.PI
                                                     0.160281533     0.0003   40.205473471    0.031681980     0.0003
                                                    -0.014806161     0.0001   40.205473471    0.031681980     0.0001
                                                    -0.000408439     0.0002   40.205473471    0.031681980     0.0002
                                                    -0.014806161     0.0006   40.130390380    0.160281533     0.0006
                                                    -0.000408439     0.0003   40.130390380    0.160281533     0.0003
                                                    -0.000408439     0.0002   40.114351399   -0.014806161     0.0002
                                              .......
```

```
2. Conversión entre coordenadas geodésicas y UTM, con cada punto en su huso.
 N.Pto        Latitud       Longitud     Coor.X-UTM     Coor.Y-UTM Huso     Kp-UTM        Conv.UTM  Latitud_DeUTM  Longitud_DeUTM    Error
 57070   40.205473471    0.031681980    249852.5848   4470607.3136   31 1.00037033    -2.119886302   40.205473471     0.031681980   0.0007
 59443   40.130390380    0.160281533    267478.0085   4455507.5880   31 1.00026561    -1.961313618   40.130390380     0.160281533   0.0004
 59341   40.114351399   -0.014806161    752811.7924   4453678.3035   30 1.00038686     2.130923241   40.114351399    -0.014806161   0.0007
 64159   39.595563014   -0.000408439    756007.3175   4431931.0363   30 1.00040692     2.142922849   39.595563014    -0.000408439   0.0007

3. Conversión entre coordenadas geodésicas y UTM, con todos los puntos en el huso 30.
 N.Pto        Latitud       Longitud     Coor.X-UTM     Coor.Y-UTM Huso     Kp-UTM        Conv.UTM  Latitud_DeUTM  Longitud_DeUTM    Error
 57070   40.205473471    0.031681980    759435.4666   4470922.4655   30 1.00042863     2.198672484   40.205473471     0.031681980   0.0008
 59443   40.130390380    0.160281533    778045.8119   4457048.1098   30 1.00055183     2.345692169   40.130390380     0.160281533   0.0014
 59341   40.114351399   -0.014806161    752811.7924   4453678.3035   30 1.00038687     2.130923241   40.114351399    -0.014806161   0.0007
 64159   39.595563014   -0.000408439    756007.3175   4431931.0363   30 1.00040693     2.142922850   39.595563014    -0.000408439   0.0007

4. Proyección de líneas geodésicas al plano UTM, utilizando el huso 30 para todas las líneas.
  N.PI  N.PF      DisLinGeo   AziDirLinGeo   AziInvLinGeo     Kd-UTM      DisUTM.Pry    DisUTM.Coor  Error.DisUTM      Conv.UTM.PI    R.AngC.PI-PF
 57070 59443     23201.6286  142.979350641  343.132210780 1.00048951      23212.9861     23212.9855       -0.5548       2.19867248    -29.00663557
 57070 59341     18464.9979  225.542238015   25.481410658 1.00040766      18472.5254     18472.5250       -0.3247       2.19867248    -34.91034514
 57070 64159     39125.4972  207.773623558    7.733621949 1.00041776      39141.8421     39141.8414       -0.6361       2.19867248    -79.26668499
 59443 59341     25446.1216  293.893422228   93.680057818 1.00046804      25458.0314     25458.0309       -0.5768       2.34569217     -7.15129536
 59443 64159     33399.0292  248.190043635   47.997787001 1.00047838      33415.0067     33415.0059       -0.7479       2.34569217    -53.51504310
 59341 64159     21972.0673  192.838625464  392.859295298 1.00039688      21980.7876     21980.7873       -0.3662       2.13092324    -43.45590889
                  Error
                 0.0007
                 0.0004
                 0.0007
                 0.0007


                  Error
                 0.0008
                 0.0014
                 0.0007
                 0.0007


                R.AngC.PI-PF     AziDirPry  AziDirUTMCoor  Error.Azi.Dir    Conv.UTM.PF   R.AngC.PF-PI      AziInvPry  AziInvUTMCoor  Error.Azi.Inv
                -29.00663557  140.78357882   140.78356465        -0.0142     2.34569217    29.68402565   340.78355021   340.78356465         0.0144
                -34.91034514  223.34705657   223.34704137        -0.0152     2.13092324    34.61069511    23.34702635    23.34704137         0.0150
                -79.26668499  205.58287774   205.58284244        -0.0353     2.14292285    78.91600020     5.58280750     5.58284244         0.0349
                 -7.15129536  291.54844519   291.54844347        -0.0017     2.13092324     6.92820825    91.54844176    91.54844347         0.0017
                -53.51504310  245.84970297   245.84968012        -0.0229     2.14292285    52.06276811    45.84965787    45.84968012         0.0222
.........       -43.45590889  190.71204781   190.71202822        -0.0196     2.14292285    43.63823401   390.71200862   390.71202822         0.0196
```

5.2.6 Salida de resultados geo_práctica5

```
SEMANA GEOMÁTICA - 2011
TALLER .......: Cálculos geodésicos y de cartografía matemática con software libre, librería GeoTop
AUTORES ......: Dr. David Hernández López, david.hernandez@ulcm.es
                Dra. Beatriz Felipe García, bfelipe@jccm.es - beatriz.felipe@uclm.es
MIGRACIÓN A PYTHON:  Juan Pedro García Tendero, juanpedrogarcia@terra.es
PRÁCTICA 5 ...: OPERACIONES DE TRANSFOMACIÓN DE COORDENADAS ENTRE LOS DATUMS ED50 Y ETRS89 EN ESPAÑA
                OPERACIÓN DE CONVERSIÓN ENTRE ALTITUDES ORTOMÉTRICAS Y ELIPSOIDALES CON UN MODELO DEL GEOIDE
- Nota.- Las coordenadas geodésicas se expresan en seudo decimal sexagesimal: gg.mmssfs
- Nota.- Las magnitudes lineales se expresan en metros.
- Sistema de Referencia ..............: ETRS89
- Parámetros del elipsoide ............: GRS80
  - Semieje mayor ....................:  6378137.000 m.
  - Inverso del aplanamiento .........:  0.003352811 adim.
  - Semieje menor ....................:  6356752.314 m.
  - Primera excentricidad al cuadrado ..:  0.006694380 adim.
  - Segunda excentricidad al cuadrado ..:  0.006739497 adim.
- Sistema de Referencia ..............: ED50
- Parámetros del elipsoide ............: INTERNACIONAL 1924
  - Semieje mayor ....................:  6378388.000 m.
  - Inverso del aplanamiento .........:  0.003367003 adim.
  - Semieje menor ....................:  6356911.946 m.
  - Primera excentricidad al cuadrado ..:  0.006722670 adim.
  - Segunda excentricidad al cuadrado ..:  0.006768170 adim.

1. Transformacion Helmert 3D 7 parametros en coordenadas cartesianas geocentricas:
   N.Punto  C.X.CG.ETRS89  C.Y.CG.ETRS89  C.Z.CG.ETRS89        Zona    C.X.CG.ED50    C.Y.CG.ED50    C.Z.CG.ED50   Latitud.ED50
         1    4594489.890    -678368.010    4357065.904    Noroeste    4594582.520    -678256.235    4357184.756   43.215585009
         3    4962848.206    -160854.365    3990884.209 R.Peninsula    4962933.906    -160749.153    3991008.649   38.584484885
        12    5103282.414    -392096.752    3793146.894 R.Peninsula    5103368.059    -391987.398    3793271.784   36.433859535
        13    4919369.704     225499.577    4039849.606  I.Baleares    4919454.661     225598.103    4039977.398   39.331370187
        15    4803054.799    -462131.609    4158378.661 R.Peninsula    4803143.686    -462025.464    4158499.727   40.564652315

   Latitud.ED50  Longitud.ED50  Alt.Elip.ED50  C.X.UTM.ED50  C.Y.UTM.ED50  Huso
   43.215585009   -8.235065568         -4.721     548826.735    4801669.099    29
   38.584484885   -1.511857075        679.030     599168.804    4315156.018    30
   36.433859535   -4.233205997         37.864     375674.184    4065600.682    30
   39.331370187    2.373234257         -1.668     467837.154    4378377.375    31
   40.564652315   -5.294022850        781.627     290018.723    4535866.608    30

2. Transformacion conforme con modelo de distorsion NTv2:
   N.Punto    Latitud.ETRS89   Longitud.ETRS89    Latitud.ED50   Longitud.ED50   C.X.UTM.ED50   C.Y.UTM.ED50  Huso  Diff.X.UTM.ED50  Diff.Y.UTM.ED50
         1     43.215177081     -8.235616747   43.215586954   -8.235064131     548827.054    4801669.701    29            0.319            0.602
         3     38.584049446     -1.512305419   38.584483584   -1.511858981     599168.350    4315155.611    30           -0.454           -0.407
        12     36.433400375     -4.233671824   36.433858703   -4.233202762     375674.983    4065600.414    30            0.799           -0.268
        13     39.330944526      2.372838313   39.331368606    2.373235495     467837.448    4378376.886    31            0.293           -0.489
        15     40.564229223     -5.294510884   40.564654074   -5.294016705     290020.175    4535867.109    30            1.452            0.501

3. Altitudes ortometricas:
   N.Punto    Latitud.ETRS89   Longitud.ETRS89   hElip.ETRS89  Ond.EGM08RedNAP   H.Ortometrica
         1     43.215177081     -8.235616747         66.917           54.426          12.491
         3     38.584049446     -1.512305419        751.736           52.050         699.686
        12     36.433400375     -4.233671824        119.848           48.085          71.763
        13     39.330944526      2.372838313         62.063           49.030          13.033
        15     40.564229223     -5.294510884        854.969           55.035         799.934
```

Con la implementación y cálculo de los 9 casos prácticos realizados e incluidos en este Proyecto Final de Máster se puede obtener una idea de la aplicabilidad de esta librería en aplicaciones de cálculos topográficos y geodésicos. Igualmente, y como uno de los puntos de partida de este Proyecto, obtenemos una librería muy potente para el uso en ámbitos universitarios, facilitando al profesorado el planteamiento y resolución de los distintos casos prácticos de las asignaturas y al tratarse de código abierto se facilita que los alumnos implementen nuevas utilidades, formatos de salida, etc. Convirtiendo geotopo en una herramienta de trabajo diario.

6 CONCLUSIONES FINALES.

Como conclusiones finales, podemos enumerar:

- Se ha corroborado lo que dicen los impulsores de Python, se trata de un lenguaje limpio, sencillo y claro, muy próximo a pseudocódigo.

- Resulta interesante la utilización de software libre para cálculos geodésicos y topografíacos. Al tratarse de software libre es accesible a cualquier tipo de usuario.

- Al partir de la consideración de un paradigma estructurado de programación, hemos conseguido implementar gran número de funciones en ficheros (módulos) de pequeño tamaño facilitando la programación. También se ha iniciado en la programación orientada a objetos con la declaración de alguna clase.

- La importación de módulos facilita a la implementación de código, sin necesitar volver a escribir el código importado y ante cualquier modificación del mismo, automáticamente se actualiza en todos los programas o módulos donde se importe dicho módulo.

- Con las funciones definidas dentro de Numpy hemos realizado la mayor parte de los cálculos necesarios en nuestra biblioteca.

- Se ha de realizar la importación expresa de gran número de operaciones matemáticas. Salvo +, -, * y /, casi todos los demás operadores se han de importar desde Numpy, por ejemplo pi, sqrt, sin, cos, etc. A veces puede resultar un poco complicado hacer la importación de estos operadores pero cuando se avanza en la programación hasta parece más didactico reflejar todas las funciones importadas de otros módulos.

- Se ha apreciado que en algunos casos, las mismas funciones están incluidas en distintos módulos y además con nombres diferentes. Así, en Numpy tenemos arcsin, arccos, arctan, etc. y en SciPy asin, acos, atan cuando se refieren a las mismas funciones arcoseno, arcocoseno y arcotangente.

- Se ha creado una librería abierta, accesible y editable para moder modificar código y también poder ampliar con el implementado de nuevas funciones que aumenten el potencial de la misma. Esto, también se incluye dentro de las líneas futuras de actuación.

7 LÍNEAS FUTURAS DE ACTUACIÓN

De forma general, se establecen las siguientes líneas futuras de actuación e investigación:

- Avance en el desarrollo a aplicaciones informáticas mediante el uso de software libre y en especial de Python.

- Estudio de la aplicabilidad de Python en el cálculo minimo-cuadrático de sistemas de ecuaciones aplicables en topografía y geodesia. Para ello se ha de profundizar en el estudio de las diferencias entre los tipos array que contiene numpy y los tipos matrix par el cálculo avanzado de sistemas de ecuaciones.

- Aplicación de Python para el cálculo y ajuste de Redes Geodésicas mediante la utilización de mínimos cuadrados. Con este fin, estaríamos evitando el tener que recurrir a los softwares de cálculo de las casas comerciales, ahorrando así un coste importante y el poder realizar los ajustes y presentaciones de resultados adapatados a nuestras necesidades.

- Inclusión de una interfaz gráfica a los cálculos realizados para poder tener visualmente el resultado de los mismos.

- Aplicabilidad de python en el procesado de imágenes. Utilización de la librería PIL. (Python Imaging Library) que agrega capacidades de procesamiento de imágenes al interprete Python.

8 ANEJO.

8.1 EQUIVALENCIAS ENTRE FUNCIONES EN MATLAB Y PYTHON.

Al partir de la libreía Geotop en Octave, hemos de tener en cuenta la equivanlencia entre algunas funciones de Matlab/Octave con Python. Así, tenemos:

MATLAB	**numpy.array**	**numpy.matrix**	**Notes**
ndims(a)	ndim(a) or a.ndim		get the number of dimensions of a (tensor rank)
size(a)	shape(a) or a.shape		get the "size" of the matrix
size(a,n)	a.shape[n-1]		get the number of elements of the nth dimension of array a. (Note that MATLAB® uses 1 based indexing while Python uses 0 based indexing, *See note 'INDEXING'*)
MATLAB	**numpy.array**	**numpy.matrix**	**Notes**
[1 2 3; 4 5 6]	array([[1.,2.,3.], [4.,5.,6.]])	mat([[1.,2.,3.], [4.,5.,6.]]) or mat("1 2 3; 4 5 6")	2x3 matrix literal
[a b; c d]	vstack([hstack([a,b]), hstack([c,d])])	bmat('a b; c d')	construct a matrix from blocks a,b,c, and d
a(end)	a[-1]	a[:,-1][0,0]	access last element in the 1xn matrix a
a(2,5)	a[1,4]		access element in second row, fifth column
a(2,:)	a[1] or a[1,:]		entire second row of a
a(1:5,:)	a[0:5] or a[:5] or a[0:5,:]		the first five rows of a
a(end-4:end,:)	a[-5:]		the last five rows of a
a(1:3,5:9)	a[0:3][:,4:9]		rows one to three and columns five to nine of a. This gives read-only access.
a([2,4,5],[1,3])	a[ix_([1,3,4],[0,2])]		rows 2,4 and 5 and columns 1 and 3. This allows the matrix to be modified, and doesn't require a regular slice.
a(3:2:21,:)	a[2:21:2,:]		every other row of a, starting with the third and going to the twenty-first
a(1:2:end,:)	a[::2,:]		every other row of a, starting with the first
a(end:-1:1,:) or flipud(a)	a[::-1,:]		a with rows in reverse order
a([1:end 1],:)	a[r_[:len(a),0]]		a with copy of the first row appended to the end
a.'	a.transpose() or a.T		transpose of a
a'	a.conj().transpose() or a.conj().T	a.H	conjugate transpose of a
a * b	dot(a,b)	a * b	matrix multiply
a .* b	a * b	multiply(a,b)	element-wise multiply

MATLAB	numpy.array	numpy.matrix	Notes
a./b	a/b		element-wise divide
a.^3	a**3	power(a,3)	element-wise exponentiation
(a>0.5)	(a>0.5)		matrix whose i,jth element is (a_ij > 0.5)
find(a>0.5)	nonzero(a>0.5)		find the indices where (a > 0.5)
a(:,find(v>0.5))	a[:,nonzero(v>0.5)[0]]	a[:,nonzero(v.A>0.5)[0]]	extract the columms of a where vector v > 0.5
a(:,find(v>0.5))	a[:,v.T>0.5]	a[:,v.T>0.5)]	extract the columms of a where column vector v > 0.5
a(a<0.5)=0	a[a<0.5]=0		a with elements less than 0.5 zeroed out
a .* (a>0.5)	a * (a>0.5)	mat(a.A * (a>0.5).A)	a with elements less than 0.5 zeroed out
a(:) = 3	a[:] = 3		set all values to the same scalar value
y=x	y = x.copy()		numpy assigns by reference
y=x(2,:)	y = x[1,:].copy()		numpy slices are by reference
y=x(:)	y = x.flatten(1)		turn array into vector (note that this forces a copy)
1:10	arange(1.,11.) or r_[1.:11.] or r_[1:10:10j]	mat(arange(1.,11.)) or r_[1.:11.,'r']	create an increasing vector *see note 'RANGES'*
0:9	arange(10.) or r_[:10.] or r_[:9:10j]	mat(arange(10.)) or r_[:10.,'r']	create an increasing vector *see note 'RANGES'*
[1:10]'	arange(1.,11.)[:, newaxis]	r_[1.:11.,'c']	create a column vector
zeros(3,4)	zeros((3,4))	mat(...)	3x4 rank-2 array full of 64-bit floating point zeros
zeros(3,4,5)	zeros((3,4,5))	mat(...)	3x4x5 rank-3 array full of 64-bit floating point zeros
ones(3,4)	ones((3,4))	mat(...)	3x4 rank-2 array full of 64-bit floating point ones
eye(3)	eye(3)	mat(...)	3x3 identity matrix
diag(a)	diag(a)	mat(...)	vector of diagonal elements of a
diag(a,0)	diag(a,0)	mat(...)	square diagonal matrix whose nonzero values are the elements of a
rand(3,4)	random.rand(3,4)	mat(...)	random 3x4 matrix
linspace(1,3,4)	linspace(1,3,4)	mat(...)	4 equally spaced samples between 1 and 3, inclusive
[x,y]=meshgrid(0:8,0:5)	mgrid[0:9.,0:6.] or meshgrid(r_[0:9.],r_[0:6.]	mat(...)	two 2D arrays: one of x values, the other of y values
	ogrid[0:9.,0:6.] or ix_(r_[0:9.],r_[0:6.]	mat(...)	the best way to eval functions on a grid
[x,y]=meshgrid([1,2,4],[2,4,5])	meshgrid([1,2,4],[2,4,5])	mat(...)	

	ix_([1,2,4],[2,4,5])	mat(...)	the best way to eval functions on a grid
MATLAB	**numpy.array**	**numpy.matrix**	**Notes**
repmat(a, m, n)	tile(a, (m, n))	mat(...)	create m by n copies of a
[a b]	concatenate((a,b),1) or hstack((a,b)) or column_stack((a,b)) or c_[a,b]	concatenate((a,b),1)	concatenate columns of a and b
[a; b]	concatenate((a,b)) or vstack((a,b)) or r_[a,b]	concatenate((a,b))	concatenate rows of a and b
max(max(a))	a.max()		maximum element of a (with ndims(a)<=2 for matlab)
max(a)	a.max(0)		maximum element of each column of matrix a
max(a,[],2)	a.max(1)		maximum element of each row of matrix a
max(a,b)	maximum(a, b)		compares a and b element-wise, and returns the maximum value from each pair
norm(v)	sqrt(dot(v,v)) or Sci.linalg.norm(v) or linalg.norm(v)	sqrt(dot(v.A,v.A)) or Sci.linalg.norm(v) or linalg.norm(v)	L2 norm of vector v
a & b	logical_and(a,b)		element-by-element AND operator (Numpy ufunc) *see note 'LOGICOPS'*
a | b	logical_or(a,b)		element-by-element OR operator (Numpy ufunc) *see note 'LOGICOPS'*
bitand(a,b)	a & b		bitwise AND operator (Python native and Numpy ufunc)
bitor(a,b)	a | b		bitwise OR operator (Python native and Numpy ufunc)
inv(a)	linalg.inv(a)		inverse of square matrix a
pinv(a)	linalg.pinv(a)		pseudo-inverse of matrix a
rank(a)	linalg.matrix_rank(a)		rank of a matrix a
a\b	linalg.solve(a,b) if a is square linalg.lstsq(a,b) otherwise		solution of a x = b for x
b/a	Solve a.T x.T = b.T instead		solution of x a = b for x
[U,S,V]=svd(a)	U, S, Vh = linalg.svd(a), V = Vh.T		singular value decomposition of a
chol(a)	linalg.cholesky(a).T		cholesky factorization of a matrix (chol(a) in matlab returns an upper triangular matrix, but linalg.cholesky(a) returns a lower triangular matrix)
[V,D]=eig(a)	D,V = linalg.eig(a)		eigenvalues and eigenvectors of a
[V,D]=eig(a,b)	V,D = Sci.linalg.eig(a,b)		eigenvalues and eigenvectors of a,b
[V,D]=eigs(a,k)			find the k largest eigenvalues and eigenvectors of a

MATLAB	**numpy.array**	**numpy.matrix**	**notes**
[Q,R,P]=qr(a,0)	Q,R = Sci.linalg.qr(a)	mat(...)	QR decomposition
[L,U,P]=lu(a)	L,U = Sci.linalg.lu(a) or LU,P=Sci.linalg.lu_factor(a)	mat(...)	LU decomposition
conjgrad	Sci.linalg.cg	mat(...)	Conjugate gradients solver
fft(a)	fft(a)	mat(...)	Fourier transform of a
ifft(a)	ifft(a)	mat(...)	inverse Fourier transform of a
sort(a)	sort(a) or a.sort()	mat(...)	sort the matrix
[b,I] = sortrows(a,i)	I = argsort(a[:,i]), b=a[I,:]		sort the rows of the matrix
regress(y,X)	linalg.lstsq(X,y)		multilinear regression
decimate(x, q)	Sci.signal.resample(x, len(x)/q)		downsample with low-pass filtering
unique(a)	unique(a)		
squeeze(a)	a.squeeze()		

9 BIBLIOGRAFÍA

DOWNEY, A.; ELKNER, J.; MEYERS, C.: *Aprenda a Pensar Como un Programador con Python.*

GARCÍA-ASENJO VILLAMAYOR, L.; HERNÁNDEZ LÓPEZ, D.: *Geodesia*. Febrero 2005

GONZÁLEZ DUQUE, R: *Python PARA TODOS.* Edición Internet bajo licencia Creative Commons Reconocimiento 2.5 España. Descargable en http://mundogeek.net/tutorial-python/

MARZAL, A., GRACIA, I.: *Introducción a la programación con Python. Departamento de Lenguajes y Sistemas Informáticos. Universitat Jaume I.* 2003. Edición Internet

NUMPY COMMUNITY.: *NumPy Reference. Release 1.6.0. Mayo.* Edición Internet. Descargable en http://www.scipy.org/

NUMPY COMMUNITY.: *NumPy User Guide. Release 1.6.0.* Mayo. Edición Internet. Descargable en http://www.scipy.org/

VAN ROSSUM, G.: *Guía de aprendizaje de Python.Release 2.4.1ª0.* Edición Intenet. Septiembre 2005. Descargable *en* http://www.scipy.org/

VAN ROSSUM, G.: *El tutorial de Python.* Edición Intenet. Descargable en http://python.org.ar/pyar/Tutorial

10 PÁGINAS WEB CONSULTADAS

http://enfoquevirtual.wordpress.com/category/tutorial/

http://geektheplanet.net/1347/how-to-instalar-pydev-en-aptana.xhtml

http://mundogeek.net/archivos/2008/03/28/python-modulos-y-paquetes/

http://www.thebitsource.com/programming-software-development/python/python-application-development-aptana-pydev/

http://www.youtube.com/watch?v=XtnfY2cBbMA.
youtube interesante trabajo con pydev y modulos

http://coding.derkeiler.com/Archive/Python/comp.lang.python/2005-08/msg03511.html
página que explica cómo solucionar los problemas de Non-ASCII character.

http://es.wikibooks.org/wiki/Inmersi%C3%B3n_en_Python/Su_primer_programa_en_Python/Documentado_de_funciones
página que explica la cadena de documentación después de definir una función con """___""". (triples comillas).

http://www.python.org/dev/peps/pep-0008/
style guide for python code.

http://mundogeek.net/archivos/2008/07/07/documentacion-en-python/
habla de la forma de documentar en python

http://www.python.org/dev/peps/pep-0257/
documentación y comentarios.

http://programandoideas.com/comentarios-en-python/
documentación y comentarios.

http://www.youtube.com/watch?v=v0sqRYuL5e8
tutorial de python con aptana.

http://www.youtube.com/watch?v=29mq1Bn52GY&feature=related
youtube tutorial

http://projects.scipy.org/numpy/browser/trunk/numpy/matlib.py?rev=8567
página que habla un poco de matrices.

http://www.gfc.edu.co/~arsran/guias42/p8guia42.html
cosas interesantes de matrices.

http://docencia-eupt.unizar.es/ctmedra/scipy_tutorial/scipy_tutorial.html
varios sobre python

http://docs.python.org/tutorial/modules.html
muy buena de paquetes, etc.

http://www.linux-itt.com/2008/04/python-mdulos-y-paquetes.html
más sobre paquetes

http://www.mclibre.org/consultar/python/otros/in03_idle.htmlç
coding cp1252.

http://www.mclibre.org/consultar/python/
tutorial de python en la web. Idle

http://bioinf.comav.upv.es/courses/linux/python/modulos.html
csv listas,etc.

http://www.davidricardo.com.mx/?p=1173
ok muy bueno importación csv como matriz

http://www.aprenderpython.org/

http://www.scipy.org/Numpy_Example_List_With_Doc
página muy importante de scipy

http://jsbsan.blogspot.com/2011/01/calculo-de-un-area-de-un-poligono.html
video sobre cálculo de areas de polígonos irregulares

http://mathesaurus.sourceforge.net/matlab-numpy.html
vip.relación entre operaciones con octave y con python

http://www.scipy.org/Tentative_NumPy_Tutorial
pagina importante de scipy

http://tecnologicum.wordpress.com/astronomia/video-tutoriales-de-python/
todos los videotutoriales de python

http://www.youtube.com/watch?v=Oj3MZXWXGiU&feature=related
csv

http://translate.google.es/translate?hl=es&langpair=en%7Ces&u=http://www.scipy.org/NumPy_for_Matlab_Users
interesante sobre numpy…

http://wiki.python.org/moin/HowTo/Sorting/
interesante de sort

http://wiki.python.org/moin/HowTo/Sorting/#Operator_Module_Functions
sort, itemgetter,etc

http://code.activestate.com/recipes/304440-sorting-dictionaries-by-value-in-python-24/
operator itemgetter

http://mundogeek.net/archivos/2008/04/02/python-entrada-salida-ficheros/
entrada salida de archivos

http://www.gulic.org/almacen/httlaclwp/chap11.htm
formato de escritura textos, números y decimales

http://www.programacion.com/articulo/guia_de_aprendizaje_de_python_65/7
formatos de salida

http://numpy.sourceforge.net/numdoc/HTML/numdoc.htm#pgfId-57315
muy importante sobre arrays,etc.

http://www.hjcb.nl/python/Arrays.html
muy importante sobre arrays,etc.

http://www.alecjacobson.com/weblog/?p=1570
cargar un txt como una matriz en python

http://docs.scipy.org/doc/numpy/reference/generated/numpy.genfromtxt.html
getfromtxt de numpy.

http://www.alegsaonline.com/art/13.php

http://www.um.es/docencia/barzana/IAGP/Iagp3.html

http://web.usal.es/~dhernand/software.htm#Geotop

http://mail.python.org/pipermail/tutor/2008-March/060886.html
From future import division.

http://www.youtube.com/watch?v=I6GMIQBD6EU
youtube diccionarios

http://mundogeek.net/archivos/2008/01/21/python-tipos-basicos-ii-colecciones/
diccionarios

http://staff.not.iac.es/~rcardenes/hg/diveintopython3-es/native-datatypes.html
diccionarios

http://www.estrellateyarde.org/discover/manual-python-lo-mas-basico
diccionarios

http://es.wikibooks.org/wiki/Inmersi%C3%B3n_en_Python/Tipos_de_datos_nativos/Presentaci%C3%B3n_de_los_diccionarios
diccionarios.

http://docs.python.org.ar/tutorial/datastructures.html#diccionarios
python estructura de diccionarios

http://www.wikilearning.com/tutorial/introduccion_informal_a_matlab_y_octave-matrices_y_algebra_lineal_i/19256-8
concatenar matrices

http://www.slideshare.net/santiagosilas/computao-cientfica-com-numpy-e-scipy-7797060
detalles de numpy, vectores, matrices,etc.

http://www.scipy.org/NumPy_for_Matlab_Users#head-e9a492daa18afcd86e84e07cd2824a9b1b651935
muy importante de scipy arrays y matrices…

http://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html
muy importante de scipy arrays y matrices…

http://pyspanishdoc.sourceforge.net/tut/
traducción del tutorial de Guido.

http://www.scipy.org/Numpy_Example_List_With_Doc#head-5202db3259f69441c695ab0efc0cdf45341829fc

http://pyspanishdoc.sourceforge.net/lib/module-pickle.html
función pickle

http://nicocesar.com/?q=node/23
carga de ficheros binarios.

http://staff.not.iac.es/~rcardenes/hg/diveintopython3-es/files.html
apertura, cierre, carga de ficheros, etc.

http://pythonr2.wordpress.com/2008/08/29/leer-archivos-binarios-en-python/
lectura de ficheros binarios.

http://www.ign.es/ign/layoutIn/herramientas.do
descarga de ficheros regilla del I.G.N.

http://terrasit.gva.es/es/kb/transformacion-coordenadas-utm-ed50-utm-etrs89-utilizando-libreria-proj4
página de la gva sobre cambio de sistema y fichero rejilla.

http://docs.python.org/library/struct.html
módulo struct de python para lectura de ficheros binarios.

http://www.slideshare.net/jpadillaa/programacion-orientada-a-objetos-en-python
programación orientada a objetos con python.

http://pyspanishdoc.sourceforge.net/tut/node7.html
recorrer diccionarios

http://www.ikeralbeniz.net/2010/12/15/conversion-de-ed50-a-wgs84-o-etrs89/
interesante, aplicación fichero rejilla ntv2 en python