Gestión de opciones y argumentos usando `docopt`

Professors de l'assignatura

noviembre de 2022

Índice

1. El interfaz en línea de comandos según POSIX y GNU **1**

1.1. Opciones y argumentos según POSIX 1

1.2. Extensiones y convenios de GNU 2

1.3. La *sinopsis* del programa 3

1.4. Ejemplo de sinopsis conforme al estándar POSIX con las ampliaciones de GNU 4

1.5. Nombre típico de los argumentos de las opciones en GNU 5

2. Gestión de opciones usando la biblioteca `docopt` **5**

2.1. Sintaxis de la sinopsis para su uso con `docopt` 6

2.2. Análisis sintáctico de la invocación 7

2.3. Acceso a los argumentos desde el programa 9

2.4. Más información de `docopt` 12

1. El interfaz en línea de comandos según POSIX y GNU

El paso de parámetros y argumentos a los programas es una cuestión de suma importancia. Existen, básicamente, dos alternativas: pasarlos en tiempo de ejecución o por línea de comandos. La primera suele requerir que el usuario introduzca texto o seleccione con el ratón o el teclado las opciones deseadas. Por lo tanto, es difícil, por no decir imposible, de automatizar. La alternativa, el uso de un *interfaz en línea de comandos* (**CLI**, por sus iniciales en inglés), es la preferida en sistemas del estilo de UNIX, y es especialmente útil para lanzar programas con la mínima interacción posible (por ejemplo, si se desea lanzar una tanda de experimentos ejecutando un mismo programa con parámetros distintos).

Desde los inicios de los sistemas informáticos han aparecido infinidad de estrategias para el paso de argumentos por línea de comandos. Siendo N el número de programadores, el número de criterios es, seguramente, no inferior a $2N$, ya que casi cada programador tiene su estilo preferido, que no le gusta a casi nadie más, y casi nunca es el primero que desarrolló.

Para evitar esta dispersión, y muchas otras que existen en el ámbito informático[1], el IEEE lanzó en los años 80 del siglo XX una iniciativa para unificar criterios, el llamado POSIX (acrónimo de *Portable Operating System Interface* rematado en la X típica de sistemas operativos de la familia del UNIX, como Linux, MacOS X, Xenix, AIX, etc.).

1.1. Opciones y argumentos según POSIX

Según POSIX, los argumentos de un programa se pueden dividir en dos tipos: opciones y operandos. A su vez, las opciones se dividen entre las que tienen un argumento y las que no. Los aspectos más relevantes de opciones y operandos son:

Opciones:
Una opción es un argumento formado por un guion (`'-'`) y un único carácter alfanumérico. Suele denominarse *nombre de la opción* al conjunto formado por ambos; así, suele decirse *opción menos hache* a la cadena `'-h'`.

- Hay dos tipos de opciones:
  - Sin argumento de la opción, también conocidas como *flags*. Su valor es booleano: `True`, si la opción está presente en la invocación; `False`, en caso contrario.
    Por ejemplo, en muchos programas la opción `'-v'` activa un modo *verboso* que muestra información más detallada de la ejecución del programa.
  - Con argumento de la opción, en cuyo caso toma el valor de su argumento, que será una sola palabra que puede estar unida o separada del nombre de la opción.
    Por ejemplo: `'-ihola.wav'` o `'-i hola.wav'`.
    Si son necesarios más de un argumento, éstos se separarán con comas o con espacios. En este último caso es muy probable que deba encerrarlos entre comillas para que el shell los interprete como una sola palabra.
- Las opciones sin argumentos pueden agruparse entre sí y con como máximo una opción con argumento, que debe aparecer en último lugar. Por ejemplo, `'-vihola.wav'` es equivalente a `'-v -i hola.wav'`
- Habitualmente, todo argumento iniciado por guion es una opción; aunque hay excepciones:

[1] El estándar ocupa 6900 páginas, de las que sólo 8 se refieren estrictamente al interfaz en línea de comandos

    - Cuando el argumento es el valor que toma una opción. Por ejemplo, si se usa la opción `'-u umbral'` para fijar un cierto umbral, en la expresión `'-u -1'`, el `'-1'` es el valor que toma el `umbral`, no una opción diferente.
    - La cadena `'--'` aislada sirve para indicar el final de las opciones. Todos los argumentos que se escriban después de ella serán interpretados como argumentos o comandos, independientemente de si empiezan por guion o no.
    - Un sólo guion aislado (`'-'`) suele usarse para indicar que un fichero de entrada debe ser sustituido por la entrada estándar (`stdin`). Si no hay ambigüedad, también puede usarse para indicar que un fichero de salida debe ser sustituido por la salida estándar (`stdout`).
  - En principio, las opciones pueden aparecer en cualquier orden, aunque cabe la posibilidad de definir situaciones concretas en las que el orden sea relevante, y que deberá gestionar el propio programa.
  - Las opciones deben anteceder al resto de argumentos. Es decir, no se pueden *mezclar* opciones y argumentos.

Operandos:

Toda cadena de texto que no pueda ser interpretada como una opción o el argumento de una opción es un operando. Son posicionales; es decir, su significado depende de la posición en que se escriben.

En la invocación, los operandos deben ser sustituidos por el valor deseado.

1.2. Extensiones y convenios de GNU

A los criterios marcados por POSIX, GNU ha añadido extensiones y convenios que se han convertido en un estándar *de facto*. Puede encontrarse información más completa en GNU Argument Syntax y en GNU: Standards for Command Line Interfaces.

Estas extensiones no están contempladas por POSIX, pero no impiden el cumplimiento del estándar en tanto en cuanto se mantenga la posibilidad de ejecutar el programa usando un CLI estrictamente POSIX. Así es habitual permitir argumentos estrictamente POSIX junto con las extensiones de GNU.

Algunas de las extensiones más importantes aportadas por GNU son:

- Nombres de opción largos: indicados por dos guiones seguidos del nombre de la opción, que será de más de un carácter. Este tipo de opción no sólo está permitido sino que, por motivos de claridad, es el más recomendado y se ha convertido en un estándar *de facto*.
    - El nombre largo deberá ser descriptivo del uso o comportamiento de la opción y, habitualmente, estará formado por una o más palabras separadas por guion (`'--opción-larga'`).
    - No es necesario escribir el nombre completo de la opción, sólo los caracteres necesarios para identificarla unívocamente.
    - Si la opción tiene un argumento, éste se escribirá separado de aquélla mediante espacios o un signo igual (`'='`).

      Por ejemplo, `'--umbral=VALOR'` es lo mismo que `'--umbral VALOR'`.
    - Suele proporcionarse una alternativa *corta*, conforme al estándar POSIX, junto a cada opción *larga*.

      Por ejemplo, si existe la opción `'--umbral=VALOR'`, será habitual que también exista la opción `'-u VALOR'`.

- Los argumentos posicionales se dividen en dos tipos: *comandos* y *operandos* propiamente dichos:
  - Un *operando* propiamente dicho es uno cuyo valor indica el usuario en la invocación.
    Por ejemplo, si un programa tiene como argumento `'fichero-entrada'`, el usuario deberá escribir el nombre de un fichero en esa posición.
  - Un *comando* es una cadena de texto que debe aparecer literalmente en la invocación.
    Por ejemplo, los programas de manejo de paquetes `apt`, en Ubuntu, o `brew`, en MacOS, usan multitud de comandos (`update`, `install`, `remove`, etc.) que indican las posibles operaciones a realizar.
  - Para distinguir comandos y operandos, los primeros se indican usando exclusivamente letras minúsculas, dígitos y subrayados(`"esto_es_un_comando"`); mientras que los operandos usarán mayúsculas, dígitos y subrayados (`"ESTO_ES_UN_OPERANDO"`), o estarán encerrados entre corchetes oblicuos (signos menor < y mayor >), en cuyo caso tienen formato libre (`"<Esto es otro operando>"`).
    Los mismos criterios de nomenclatura se aplican a los argumentos de las opciones.
- Salvo que se indique lo contrario, opciones y argumentos pueden aparecer en cualquier orden. Es decir, no es necesario que las opciones aparezcan antes de los argumentos.
- Las opciones `-h`, `--help` y `--version` suelen tener un significado especial: las dos primeras muestran la sinopsis del programa, la última su versión.

1.3. La *sinopsis* del programa

Todo programa debe contar con una *sinopsis* que indique su modo de empleo. Aunque POSIX no lo especifica, las opciones `'-?'` o `'-h'` suelen ser indicativa de que el usuario quiere ver esta sinopsis en pantalla. En los programas que usan las extensiones de GNU, las opciones `'-h'` o `'--help'` cumplen esta misión.

Los elementos más relevantes de la sinopsis son:

- Si un programa tiene distintos modos de operación, con opciones diferentes, cada modo se escribe en una línea separada.

  A menudo, los programas tienen tres modos de operación: el modo normal, en el cual se ejecuta el programa para que realice el cometido para el que se ha escrito; el modo de ayuda, que muestra un mensaje en pantalla informando de cómo se ha de invocar; y el modo versión, que muestra la versión del programa y finaliza la ejecución.
- Los argumentos de valor variable pueden escribirse usando sólo mayúsculas, dígitos y subrayados (`'ESTO_POR_EJEMPLO'`) o encerrados entre corchetes oblicuos (`'<Esto otro>'`).

  Toda cadena compuesta exclusivamente de minúsculas, dígitos y subrayados se interpreta como un comando, y debe escribirse tal cual en la invocación. Por ejemplo, en la orden `'sudo apt install ...'`, la palabra `'install'` es un comando que debe escribirse literalmente.
- Los argumentos opcionales se escriben encerrados entre corchetes (`'[argumento-opcional]'`). Por ejemplo, `'[-v]'` es un argumento opcional, que puede especificarse o no.
- Los argumentos obligatorios pueden aparecer tal cual o encerrados entre paréntesis. Por ejemplo, `'(<argumento-obligatorio>)'` o, simplemente, `'<argumento-obligatorio>'`.

  El uso de los paréntesis sólo suele ser conveniente para agrupar argumentos alternativos o que pueden repetirse más de una vez.

- Los argumentos alternativos, es decir, mutuamente excluyentes, se escriben separados por barra vertical (`'|'`) y, a menudo, encerrados entre corchetes si son opcionales, o entre paréntesis si son obligatorios.
- Cuando un argumento puede ser repetido un número indefinido de veces, se indica con tres puntos (`elipsis` o `'...'`) a continuación del argumento correspondiente.

  El operador elipsis es *unario* y sólo afecta al argumento o grupo de argumentos precedente. Para que afecte a más de un argumento, todo el conjunto debe estar encerrado entre paréntesis o corchetes.

1.4. Ejemplo de sinopsis conforme al estándar POSIX con las ampliaciones de GNU

El recuadro siguiente muestra una sinopsis típica siguiendo las recomendaciones de POSIX con las extensiones de GNU. Es un ejemplo clásico, útil para ilustrar (casi) todas las posibilidades comentadas en las dos secciones precedentes:

```
Naval Fate.

Usage:
  naval_fate ship new <name>...
  naval_fate ship <name> move <x> <y> [--speed=<kn>]
  naval_fate ship shoot <x> <y>
  naval_fate mine (set|remove) <x> <y> [--moored|--drifting]
  naval_fate -h | --help
  naval_fate --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --speed=<kn>  Speed in knots [default: 10].
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.
```

El ejemplo muestra las opciones de un juego de barcos, en el que el usuario puede manejar barcos (*ship*) o minas (*mine*). Los argumentos que no están rodeados por corchetes oblicuos son considerados literales, y han de aparecer tal cual se escriben. En función de qué acción se desea realizar, pueden usarse unas opciones u otras.

Por ejemplo, para crear el vapor *el Català* y el bajel pirata *el Temido*, y desplazar el primero a La Habana (coordenadas `x, y=100, 230`) con una velocidad de 40 nudos, ejecutaríamos el programa con los siguientes argumentos:

```
usuario:~/naval$ naval_fate ship new el-catala el-temido
usuario:~/naval$ naval_fate ship el-catala move 100 230 --speed=40
```

1.5. Nombre típico de los argumentos de las opciones en GNU

En la sinopsis del ejemplo, el argumento de la opción `'--speed'` recibe el nombre `'<kn>'`, abreviatura de *knot* (nudo, en inglés). Está claro que se trata de un número, pero no se sabe si tiene que ser entero o real. El valor por defecto, 10, tampoco ayuda mucho; ¿podemos usar un valor no entero?

Lo cierto es que usar el nombre `'<kn>'` no es de gran ayuda, ya que el significado del argumento aparece en la descripción de la opción, y además en formato completo, no como una abreviatura que se puede entender o no. Hubiera sido más interesante si se nos informara del tipo de argumento esperado.

El convenio usado por GNU en sus programas es muy recomendable: consiste en indicar el tipo esperado del argumento usando una palabra en mayúsculas. Por ejemplo, en el caso de la velocidad del barco, podríamos usar `'INT'`, `'REAL'`, `'FLOAT'`, etc.

Algunos nombres típicos usados por GNU son los siguientes, aunque no es necesario adherirse a ellos de manera estricta, siempre que su significado quede claro al usuario:

`'INT'`: Número entero. En ocasiones, se usa alguna expresión de carácter inequívocamente entero, como `'SIZE'` o `'LENGTH'`.

`'REAL'`: Número real. Otras posibilidades son `'FLOAT'`, `'DOUBLE'`, etc.

`'CHAR'`: Un único carácter; a veces abreviado como `'CHR'`. En ocasiones se puede indicar el tipo concreto de carácter: `'LETTER'`, `'DIGIT'`, etc.

`'STRING'`: Cadena de caracteres; a veces abreviado como `'STR'`.

En ocasiones, se usa un nombre específico que indica el tipo de cadena esperado: `'WORD'`, `'SENTENCE'`, `'NAME'`, `'CITY'`, etc.

`'FILE'`: Nombre de fichero. También se usa `'FILENAME'`.

2. Gestión de opciones usando la biblioteca `docopt`

Para facilitar la gestión del CLI se han desarrollado toda una serie de aplicaciones y funciones de librería. Por ejemplo, durante mucho tiempo la única opción en programas C era la función `getopt()` (ejecutar `man 3 getopt` para ver su funcionamiento).

El problema con estas soluciones es que todo el trabajo de gestión de las opciones y del mensaje con la sintaxis recae sobre el programador. Eso quiere decir que es él el que debe garantizar que el mensaje mostrado en pantalla se corresponde exactamente con la funcionalidad del programa. Por desgracia, es habitual que las modificaciones en esta última no se reflejen adecuadamente en aquél. Así, es frecuente encontrar en los mensajes de sinopsis gestionados de este modo opciones obsoletas (que ya no son operativas o han cambiado su significado) u ocultas (que no se han incorporado al mensaje).

En el lenguaje Python, se introdujo la función `optparse()`, luego sustituida por `argparse()`. Además de ser notoriamente más versátiles y potentes que el `getopt()` de C, la gran ventaja de `optparse()` es que, a partir de la definición algorítmica de las opciones del programa, el mensaje de sinopsis se genera automáticamente. No obstante, la gestión de las opciones continuaba siendo bastante farragosa (aunque no tanto como con `getopt()`), y no proporciona demasiada flexibilidad en la confección del mensaje de sinopsis.

En 2012, Vladimir Keleshev ideó un sistema de gestión del CLI para programas escritos en Python muy original: dado que todo programa debería tener un mensaje de sinopsis que, a su vez, sirve de documentación del código, construyamos un mecanismo que permita realizar el análisis de los argumentos

y opciones a partir del mismo. Esta estrategia tiene dos ventajas: permite mayor flexibilidad en la construcción del mensaje de sinopsis y, sobre todo, la gestión algorítmica de las opciones se simplifica al extremo.

En su versión inicial, esta idea se implementó como el módulo de Python `docopt`, aunque luego se ha adaptado a otros lenguajes, como C/C++, Java, Ruby, etc.

Puede encontrar más información acerca de `docopt`, su modo de empleo y todas sus opciones en su repositorio de GitHub.

2.1. Sintaxis de la sinopsis para su uso con `docopt`

Para ser interpretado correctamente por `docopt`, el mensaje con la sinopsis debe cumplir una serie de condiciones. Tomaremos el ejemplo en la página 4 para explicar cada una de ellas:

- Las primeras líneas son opcionales y sirven para describir, en pocas palabras, el cometido del programa. El estándar POSIX recomienda que esta descripción sea lo más concisa posible y que no repita innecesariamente el nombre del programa.

  En el ejemplo, la primera línea es, simplemente, `'Naval Fate'`.

- Una sección de modo de empleo encabezada por la cadena `'Usage:'` (sin importar minúsculas y mayúsculas) y rematada por una línea en blanco.

  - La sección puede contener tantas líneas de modo de empleo como modos distintos permita el programa. Cada modo de empleo debe empezar por el nombre del programa seguido por las opciones y argumentos que lo distinguen del resto de modos.

    En el ejemplo, tenemos modos de empleo diferentes para la creación de barcos, su movimiento, su disparo, etc.

    - Los modos especiales `-h | --help` y `--version` muestran el mensaje de sinopsis y la versión de programa, respectivamente, y finalizan la ejecución. En principio, de esta gestión se encarga el propio `docopt`.
    - Habitualmente, los programas sólo tendrán un modo de empleo principal. Sin embargo, es conveniente mostrar las opciones `-h | --help` y `--version` como modos de empleo diferenciados, ya que es la información que recibirá en primera instancia el usuario si invoca el programa con opciones o argumentos incorrectos.
  - Cada modo de empleo debe incluir todos sus argumentos posicionales, indicando su carácter opcional, obligatorio, alternativo o repetitivo conforme a lo indicado en el apartado 1.3.
  - Las opciones del programa también pueden indicarse en el modo de empleo. Aunque las que son realmente opcionales pueden agruparse en la cadena `'[options]'` y posponer su descripción hasta la sección `'Options'` de la sinopsis.

- Una sección de descripción de las opciones encabezada por la cadena `Options:` (o cualquier otro nombre).

  - Todas las líneas de la sinopsis que empiecen por guion (sin tener en cuenta espacios o tabuladores), son consideradas el inicio de la descripción de una opción. Esta descripción puede abarcar más de una línea, incluyendo líneas en blanco.
  - Todas las opciones del programa pueden ser descritas en esta sección, aunque no todas deben estarlo.

    En los casos siguientes, la inclusión de la opción sí es necesaria:

    - Si la opción está englobada en la cadena `'[options]'` del modo de empleo. Es decir, si no se incluye explícitamente en los modos de empleo.
    - Si se desea dar un valor por defecto al argumento de la opción. En ese caso, el valor se incluirá en la descripción en la forma `'[default: valor-por-defecto]'`
  - Si una opción puede escribirse en su forma corta o larga, ambas se escribirán en la misma línea, separadas por un único espacio o una coma.
  - Entre la opción, junto a su posible argumento, y la descripción debe haber un mínimo de dos caracteres en blanco.

    Estos dos espacios en blanco son muy importantes, ya que el mecanismo usado por `docopt` para separar la opción de su descripción se basa en ellos. Si no colocamos dos espacios entre ambas, `docopt` se armará un lío; si colocamos dos espacios en medio de la propia opción (por ejemplo, entre la opción y su argumento, o entre la opción larga y la corta), también.

- Una sección opcional final de formato libre y que se puede usar para describir el funcionamiento del programa, su autoría y/o copyright o cualquier otra cosa que se considere de interés.

  Es conveniente, por ejemplo, incluir una sección de nombre `'Argumentos:'`, o semejante, donde se explique el significado de los argumentos posicionales del programa.

2.2. Análisis sintáctico de la invocación

En Python, el análisis de la invocación se realiza con la función `docopt.docopt`, cuya sintaxis es:

```
docopt.docopt(sinopsis, argv=None, help=True, version=None, options_first=False)
```

Donde:

`sinopsis`: Cadena de texto con el mensaje de la sinopsis. A menudo, es el *docstring* del fichero (`__doc__`), aunque puede ser cualquier otra cadena.

Es el único argumento no inicializado y, por tanto, obligatorio.

`argv`: Argumentos a analizar en la forma de lista de cadenas de texto. Por ejemplo, en el caso de Naval Fate podríamos pasar `argv=['ship', 'new', 'el-catala']` para crear el barco a vapor *El Català*.

Si se usa la opción por defecto, `argv=None`, se emplea la lista de argumentos usados en la invocación del programa (`sys.argv[1:]`).

`help=True`: Indica a la función que, en caso de encontrar la opción `'-h'` o `'--help'`, debe mostrar el mensaje de sinopsis en pantalla y finalizar la ejecución.

Si se prefiere realizar la gestión de la ayuda de otro modo, deberá especificarse `help=False`.

`version=None`:
Objeto imprimible, habitualmente una cadena de texto, que se mostrará en pantalla si se ejecuta el programa con la opción `'--version'`. Si es distinta de `None`, `docopt()` lo mostrará en pantalla y finalizará la ejecución.

`options_first=False`:
Indica si el programa debe seguir el criterio POSIX estricto que requiere que las opciones vayan antes de cualquier argumento posicional.

Al llamar al programa desde el shell, lo primero que hace `docopt` es analizar sintácticamente la orden invocada para determinar si es correcta o si se han usado las opciones especiales `--help` o `--version`. Para que `docopt()` maneje estas opciones especiales, además de su uso en la invocación de la función, deberán aparecer listadas en el mensaje de sinopsis.

Si la invocación incluye la opción `--help`, `docopt()` mostrará el mensaje de sinopsis entero y acabará la ejecución del programa:

```
usuario:~/naval$ naval_fate --help
Naval Fate.

Usage:
  naval_fate ship new <name>...
  naval_fate ship <name> move <x> <y> [--speed=<kn>]
  naval_fate ship shoot <x> <y>
  naval_fate mine (set|remove) <x> <y> [--moored|--drifting]
  naval_fate -h | --help
  naval_fate --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --speed=<kn>  Speed in knots [default: 10].
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.
```

Si la invocación incluye la opción `--versión` (y no incluye la `--help`), se muestra la cadena de texto pasada a la función `docopt.docopt()` como argumento `version`. Por ejemplo, si al invocar a la función desde el programa, se hubiera hecho con la opción `docopt.docopt(..., version='Naval Fate v1.0', ...)`, el resultado sería:

```
usuario:~/naval$ naval_fate --version
Naval Fate v1.0
```

Si la invocación no incluye ninguna de las dos opciones especiales, pero tiene algún error de sintaxis, `docopt` muestra los modos de empleo enumerados en la sección `'Usage'`.

Típicamente, aunque no siempre, los programas requieren de una opción o argumento como mínimo. Así que invocar el nombre del programa *a secas* constituye un error que provocará que se muestre el modo o modos de empleo correctos. Ésta es una manera muy usada de ver cómo hay que invocar un programa (aunque, a veces, se nos puede quedar cara de tonto al hacerlo; pruebe ejecutando `cat` o `echo` sin ningún argumento para ver ejemplos de ello):

```
usuario:~/naval$ naval_fate
Usage:
  naval_fate ship new <name>...
  naval_fate ship <name> move <x> <y> [--speed=<kn>]
  naval_fate ship shoot <x> <y>
  naval_fate mine (set|remove) <x> <y> [--moored|--drifting]
```

```
naval_fate -h | --help
naval_fate --version
```

Finalmente, si no hay ningún error sintáctico y no se han usado las opciones especiales `--help` o `--version`, `docopt()` no mostrará nada, analizará el contenido de opciones y argumentos y proseguirá la ejecución del programa.

2.3. Acceso a los argumentos desde el programa

Una vez escrito el mensaje de sinopsis tal cual se desea que aparezca en pantalla, éste se pasa la función `docopt()` que, al ejecutar el programa, devolverá un diccionario en el que las claves son los nombres de los argumentos u opciones (en su versión larga, si ésta existe), y los valores son las cadenas de texto pasadas como argumento (que deberán interpretarse y convertirse al formato deseado usando las funciones apropiadas).

En el diccionario devuelto por `docopt()` los valores almacenados tienen el tipo siguiente:

Comandos y *flags* simples.
: Los comandos y *flags* que sólo pueden aparecer una vez se almacenan como variables de tipo `bool`, que puede tomar el valor `True` si son nombrados en la línea de comandos, o `False`, si no lo son.

Comandos y *flags* repetibles.
: Los comandos y *flags* que pueden aparecer más de una vez toman valores enteros que indican cuántas veces lo hacen.

Argumentos repetibles.
: Si un argumento puede ser repetido, los distintos valores se almacenan en una lista de cadenas de texto. Si no se indica el argumento, la lista estará vacía; si sólo se indica una vez, la lista sólo tendrá un elemento. En todos los casos, el valor devuelto es una lista.

Resto de argumentos.
: En el resto de casos, el valor del argumento se pasa como una cadena de texto, tal cual se escribió en la invocación, o como el objeto `None`, si no se escribió su valor.

  Es responsabilidad del programador el comprobar si el argumento es `None` o una cadena de texto y, en este último caso, interpretar correctamente su valor.

Valores por defecto.
: En argumentos repetibles, el valor por defecto se interpretará como una lista de palabras separadas por espacios, incluso en el caso en que haya una sola palabra.

  En argumentos que sólo pueden aparecer una vez, se interpretará como una única cadena de texto, aún en el caso de que esté formada por distintas palabras.

  En comandos y flags no tiene sentido usar valores por defecto.

Por ejemplo, el código siguiente analizaría el mensaje de sinopsis en la página 4 y, en el caso de que la acción fuera `move`, extraería el nombre del barco (argumento `<name>`) y el valor de la velocidad (argumento de la opción `--speed`):

```
from docopt import docopt

if __name__ == '__main__':
    arguments = docopt(sinopsis, version='Naval Fate 2.0')

    if arguments['move']:
        name = arguments['<name>'][0]
        speed = float(arguments['--speed']) if arguments['--speed'] else None

    ...
```

Fijémonos en que, al acceder al diccionario devuelto por `docopt()`, las claves han de pasarse exactamente como se han escrito en la sinopsis, incluyendo los corchetes triangulares (`'<name>'`) o los guiones (`'--speed'`).

En el caso de `arguments['<name>']`, como en el comando `'new'` el número de nombres es variable, su valor se pasa como lista de cadenas de texto, aunque en el propio comando `move` se trate de sólo uno. El comportamiento de `docopt` en estos casos es devolver el caso más general posible, que es la lista de cadenas. Por tanto, debemos acceder al primer (y único) elemento de la lista[2].

La velocidad, argumento de `--speed` nunca puede ser `None`, ya que en la sinopsis se le da el valor por defecto `speed = 10`. No obstante, es conveniente habituarse a comprobar el valor de los argumentos antes de pasárselos a una función como el constructor `float()`, ya que, en caso de que fuera `None` se produciría un error.

Nótese también la necesidad de convertir el argumento de la opción `--speed` a real; los valores devueltos por `docopt` son de alguno de los tipos enumerados en la página anterior, en general, cadenas de texto o listas de cadenas. Es responsabilidad del programador convertirlos al tipo deseado.

Ejemplo completo de la sinopsis de Naval Fate

El código completo de una versión del programa Naval Fate podría tener la forma siguiente:

```
#! /usr/bin/python3

import sys
from docopt import docopt

def crea_barco(name):
    print(f'Cremos el barco "{name}"')

def mueve_barco(name, x, y, speed):
    print(f'Enviamos el barco "{name}" a las coordenadas {x=}, {y=} ', end='')
    print(f'con una velocidad {speed=} nudos')

def dispara(x, y):
    print(f'Disparamos a las coordenadas {x=}, {y=}')
```

[2]Este es un caso un poco especial, en el que el comportamiento de `docopt` es completamente arbitrario. Existirían otros posibles comportamientos, como que se usara una lista de cadenas en un caso y una cadena en el otro, o que se produjera un error. En general, no será necesario tener en cuenta estos detalles.

```
15
16  def mina(x, y, set_, moored, drifting):
17      print(f'{"Colocamos" if set_ else "Retiramos"} una mina', end='')
18      print(f'{" fija" if moored else " a la deriva" if drifting else ""}', end='')
19      print(f' en las coordenadas {x=}, {y=}')
20
21
22  sinopsis = f"""
23      Naval Fate.
24
25      Usage:
26          {sys.argv[0]} ship new <name>...
27          {sys.argv[0]} ship <name> move <x> <y> [--speed=<kn>]
28          {sys.argv[0]} ship shoot <x> <y>
29          {sys.argv[0]} mine (set|remove) <x> <y> [--moored|--drifting]
30          {sys.argv[0]} -h | --help
31          {sys.argv[0]} --version
32
33      Options:
34          -h --help     Show this screen.
35          --version     Show version.
36          --speed=<kn>  Speed in knots [default: 10].
37          --moored      Moored (anchored) mine.
38          --drifting    Drifting mine.
39  """
40
41  if __name__ == '__main__':
42      args = docopt(sinopsis, version='Naval Fate 2.0')
43
44      # Si se han indicado las coordenadas <x> e <y>, usadas en tres de los
45      # cuatro modos, las convertimos a real.
46      try:
47          x = float(args['<x>']) if args['<x>'] else -1
48          y = float(args['<y>']) if args['<y>'] else -1
49      except:
50          print('Las coordenadas "<x>" e "<y>" deben ser números reales')
51          exit(-1)
52
53      if args['ship']:
54          if args['new']:
55              for name in args['<name>']:
56                  crea_barco(name)
57          elif args['move']:
58              try:
59                  speed = float(args['--speed'])
60              except:
61                  print('La velocidad debe ser un número real')
62                  exit(-1)
63
64              mueve_barco(args['<name>'][0], x, y, speed)
```

```
65          elif args['shoot']:
66              dispara(x, y)
67      elif args['mine']:
68          # Tal y como se ha escrito la sinopsis, 'set' y 'remove' son mutuamente
69          # excluyentes y se ha de indicar uno u otro. Por tanto, si 'set' es True,
70          # 'remove' es False y viceversa.
71          #
72          # En el caso de '--moored' y '--drifting', también son mutuamente
73          # excluyentes, pero no son obligatorios. Por tanto, los dos podrían ser
74          # False simultáneamente. Seguramente es un error del autor del paquete,
75          # porque mucho sentido no tiene.
76          mina(x, y, set_=args['set'], moored=args['--moored'], drifting=args['--drifting'])
```

2.4. Más información de `docopt`

La documentación completa de `docopt`, con una explicación de su uso, la descripción de todas sus opciones y varios ejemplos muy ilustrativos se encuentra en su repositorio de GitHub. Es muy recomendable leer la información en él contenida.