GENERADOR DE BENCHMARKS DE MÁQUINAS DE ESTADOS

Trabajo fin de grado

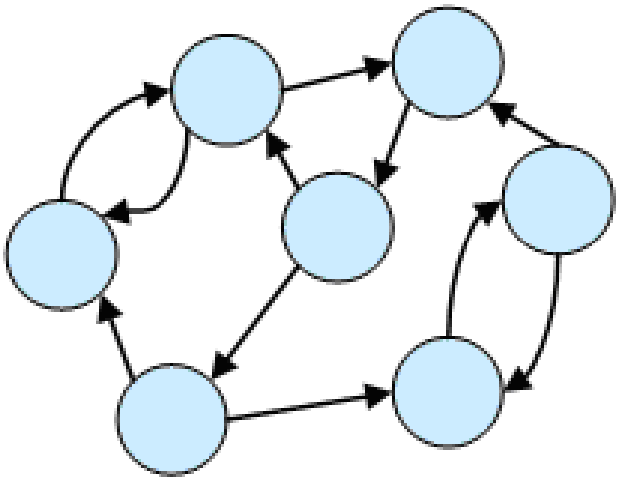
El proyecto consiste en desarrollar una aplicación que genere especificaciones de máquinas de estados para verificar y evaluar técnicas de implementación.

ALUMNO

Segura Cano, Antonio

TUTOR

Senhadji Navarro, Raouf



Escuela Técnica Superior de Ingeniería Informática

ETSII

Índice

Licencias 3
Creative Commons 3
MIT 4
Prefacio 5
¿Por qué la necesidad de esta herramienta? 5
Integrantes 6
Herramientas 7
IDE – PyCharm 7
Bibliotecas 9
Dot 9
NumPy 10
GIT 11
Entendiendo su estructura y funcionamiento 12
Comandos básicos usados 14
GitHub 15
Lenguaje Python 18
Introducción 18
Tipos de datos 18
Funciones 21
Estructuras de control 22
PIP 24
Pep 8 25
Tkinter 27
Proyecto FSpyChine 31
Especifiones 31
Instalación 31
Instalando python2 31
Instalando graphviz dot 33
Instalación de git (opcional) 34

Convención de versiones 35
¿Cuándo pasar a la versión 1? 35
TDD 36
¿Cuál es el objetivo principal de TDD? 36
Formato Kiss2 37
Clase FSM 38
Estructura de objetos 38
Funciones 39
Random 39
Patrones 40
Secuencial 40
Python DOC – FSM 42
Cómo ejecutar el proyecto 43
Con git 43
Sin git 43
Ejemplos de ejecución 44
Bibliografía 47
Terminología 48

LICENCIAS

En este apartado se especificarán los tipos de licencias por las cuales son protegidos los derechos del autor y su trabajo.

Creative Commons

Esta licencia (1) proviene de una organización sin ánimo de lucro que permite compartir y usar la creatividad y el conocimiento a través de herramientas legales. El tipo de licencia sobre la cual se basa todo el trabajo es la siguiente:

CC BY-SA

Esta licencia permite a otros re-mezclar, modificar y desarrollar sobre tu obra incluso para propósitos comerciales, siempre que te atribuyan el crédito y licencien sus nuevas obras bajo idénticos términos. Esta licencia es a menuda comparada con las licencias de "copyleft" y las de software "open source". Cualquier obra nueva basada en la tuya , lo será bajo la misma licencia, de modo que cualquier obra derivada permitirá también su uso comercial. Esta licencia es la utilizada por Wikipedia y se recomienda para aquellos materiales que puedan beneficiarse de la incorporación de contenido proveniente de Wiikipedia u otros proyectos licenciados de la misma forma.

Reconocimiento

Debe reconocer la autoría de la obra de la manera especificada por el autor o el licenciador.

Compartir bajo la misma licencia

El licenciador le permite distribuir obras derivadas sólo bajo una licencia idéntica a la que regula su obra o una licencia compatible.

MIT

Es otorgada por el Instituto de Tecnología de Massachusetts (2) y es una licencia de software gratuita.

Copyright (c) 2015 Universidad de Sevilla

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Con esta licencia conseguimos:

1. Mantener nuestro reconocimiento y copyright en el caso de que un tercero quiera manipular el software.

2. Proporcionar una serie de privilegios tales como el libre uso de la herramienta y de la copia y modificación del mismo.

3. A temas legales avisamos de que el software proporcionado no está sujeto a ningún tipo de garantía. (3)

PREFACIO

¿Por qué la necesidad de esta herramienta?

Actualmente con el reciente avance tecnológico, y sobretodo el del campo de la microelectrónica, que nos permite introducir un sistema complejo en un chip diminuto, se han ramificado y multiplicado las utilidades de los procesadores. Ordenadores, relojes, móviles e incluso electrodomésticos contienen diversos procesadores para llevar a cabo sus funcionalidades y facilitarnos la vida de una manera u otra.

Sin embargo, con las exigencias que pedimos y la multitud de necesidades que poseemos, queremos máquinas específicas diseñadas para un fin concreto. Esta realidad hace que tengamos que generalizar los circuitos integrados, y por tanto, tengamos ineficiencias ya que estamos sacrificando parte de la eficiencia y ganando en tiempos.

Las oportunidades creadas por la microelectrónica moderna no pueden efectivamente ser explotadas sin métodos adecuados y de herramientas de automatización de diseño electrónico (EDA para el diseño de circuitos y sistemas de alta calidad. Por lo tanto, la calidad de los métodos y herramientas EDA, así como de los métodos y medios para su análisis de la calidad es de vital importancia.

Dos enfoques básicos se utilizan para estudiar el rendimiento de los objetos (por ejemplo, métodos o herramientas): analítica y experimental. El enfoque analítico se basa en modelos (matemáticos) y el uso del análisis lógico y métodos de prueba. El enfoque experimental se basa en la realización de series de pruebas, la recolección y procesamiento de datos de las pruebas, y sacar conclusiones lógicas a partir de los datos recogidos y tratados.

Para llevar a cabo todos estos estudios es conveniente saber lo que queremos y cómo queremos abarcarlo. Para empezar existen dos tipos de enfoque básicos: el enfoque analítico y el experimental.

- **Analítico**: basado en modelos matemáticos y el uso de la lógica junto a una batería de pruebas.

- **Experimental**: se basa en la realización de pruebas y recolección de datos para posteriormente procesarlas y sacar conclusiones lógicas a partir de los datos recogidos.

Este trabajo está fuertemente vinculado con el análisis de rendimiento experimental de los métodos de diseño de circuitos digitales utilizado en las herramientas EDA.

La técnica de Benchmarking es el proceso mediante el cual, utilizando medidores, variables y puntos de referencia, se miden diversos aspectos para analizar, evaluar y comparar una determinada máquina con el resto.

Nuestra herramienta podrá, en términos generales, generar máquinas de estados para facilitarle los benchmarks a la comunidad encargada de esta labor. Pudiendo asímismo generar diversos tipos de máquinas muy parametrizables e incluso aportando al resto de la comunidad sus algoritmos para la generación de otros tipos de máquinas de estado.

Integrantes

Los integrantes de este trabajo son:

TUTOR: Raouf Senhadji Navarro

Doctor en Informática Industrial por la Universidad de Sevilla. Actualmente es Profesor Contratado Doctor del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla (España). Ha participado en más de 10 proyectos de investigación y publicado más de 30 artículos científicos en revistas y congresos. Sus temas de investigación incluyen síntesis lógica, implementación de máquinas de estados en FPGA, computación reconfigurable, algoritmos de optimización combinatoria y sistemas difusos. Ha pertenecido al comité científico de diversos congresos internacionales y es revisor habitual de varias revistas científicas internacionales. (4)

ALUMNO: Antonio Segura Cano

Estudiante de la rama de Ingeniería informática (Tecnologías Informáticas), desarrollador de aplicaciones móviles para Android e iOS, responsable de la seguridad y coworker en la empresa Elelog SL.

Creador del proyecto FSMpychine.

HERRAMIENTAS

En esta sección hablaremos de todas y cada una de las herramientas que nos han hecho posible la realización del trabajo.

IDE – PyCharm

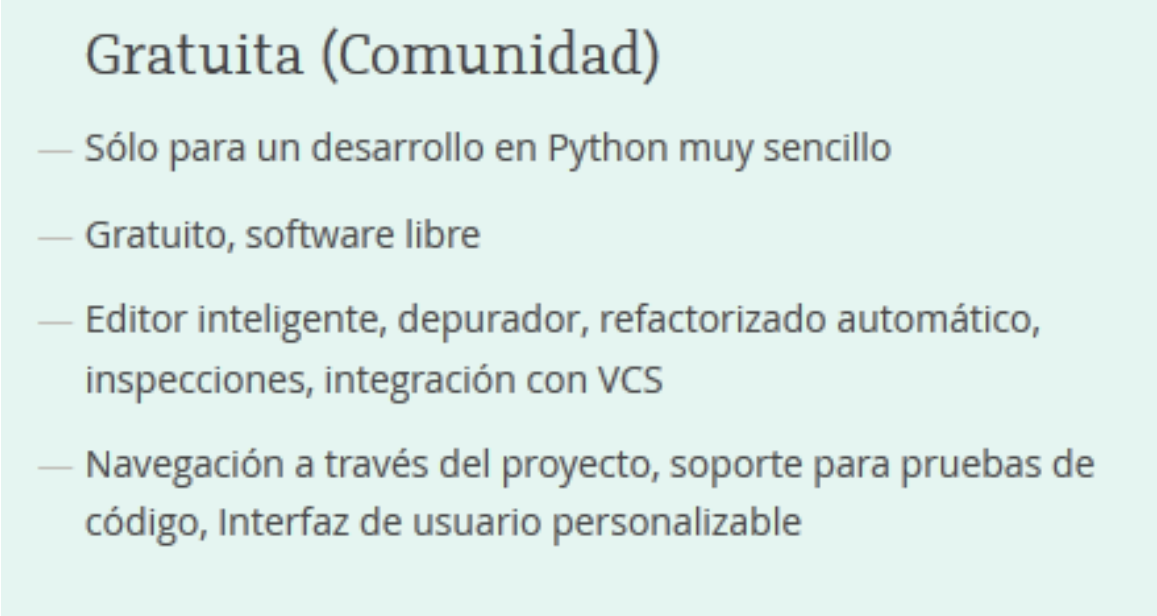
es una herramienta creada por JetBrains (5) que nos ofrece un entorno de desarrollo integrado (IDE) para escribir código especialmente en Python. Ofrece otros tipos de lenguajes y estandars como reconocimiento de documentos en formato XML o plugins para los **gitignore,** JavaScript, CoffeeScript, TypeScript y todas las tecnologías Web ya que podemos integrar también nuestros proyectos Django con ello.

Dicha herramienta posee dos versiones: la profesional y la de la comunidad.

- **Profesional:** es la versión de pago que nos facilita JetBrains, si nos adentramos en su página Web podemos ver que nos ofrece esta herramienta con todas estas características:

> Edición profesional
>
> — IDE para Python & Desarrollo Web completo
>
> — Da soporte a Django, Flask, Google App Engine, Pyramid, web2py
>
> — JavaScript, CoffeeScript, TypeScript, CSS, Cython, Lenguajes de plantillas y mucho más
>
> — Desarrollo remoto, Python Profiler, Bases de datos y asistente SQL, además de un soporte para diagramas UML y SQLAlchemy

- **Gratuita:** la versión gratuita es la ofrecida por la comunidad, es libre y no contiene tantas ayudas para fomentar la productividad como las que tiene PyCharm Profesional.



Todas estas características y diferencias entre las dos versiones las podemos encontrar en la asiguiente cita, donde se hace una comparación entre ambas mucho más detallada. (6)

En este trabajo nos hemos decantado por la versión gratuita, ya que como queremos desarrollar el uso de una herramienta libre, fomentaremos todas las demás herramientas que nos ayuden con software libre.

Bibliotecas

Dot

Dot es una herramienta que dibuja tanto grafos como árboles de jerarquías. Puede ejecutarse en línea de comandos, en un servidor Web o con una interfaz gráfica compatible.

Una de las características por las cuales destaca esta herramienta es por los algoritmos que tiene para ubicar nodos, aristas, etiquetas...

Como hemos comentado anteriormente, podemos dibujar grafos (especialmente dirijidos) y luego exportarlos como formatos de grafo o formato ilustrativo tales como GIF, PNG, SVG, PDF...

Para que nos hagamos una idea de cómo funciona dot, procedamos a explicar brevemente su funcionamiento. Dot dibuja grafos en cuatro fases principales:

1. El primer paso es tratar de romper cualquier ciclo mediante la inversión de la dirección interna para generar las aristas cíclicas.

2. El siguiente paso asigna nodos a filas discretas o niveles. En un dibujo de arriba a abajo, las filas determinan las coordenadas. Los bordes que abarcan más de un rango se rompen en las cadenas de nodos "virtuales" y bordes en unidad de longitud.

3. El tercer paso es el encargado del visualizado fetén, ya que es el responsable de que haya el número mínimo de cruces en el grafo. Lo que viene siendo en resumen, si el grafo a dibujar es plano, lo hará.

4. El último paso es reestablecer las coordenadas de los nodos para mantener las aristas lo más cortas posibles.

Nosotros la utilizaremos para generar una representación visual de la máquina de estados generada.

NumPy

NumPy es un paquete especialmente para Python cuyos fines son puramente científicos, sus principales características por las cuales hacen de él una herramienta científica imprescindible son:

- Tiene un objeto de arrays de orden n optimizado.
- Poderosas y sofisticadas funciones.
- Herramientas para integrar con otros lenguajes tales como C++ y Fortran.
- Álgebra lineal, transformaciones de Fourier y opciones de creado de números aleatorios.

En nuestro trabajo hemos elegido esta biblioteca para la funcionalidad de números aleatorios, ya que nos puede generar todo tipo de números aleatorios satisfaciendo todas nuestras necesidades.

El porqué de utilizarla en el proyecto es porque si queremos generar una máquina de estados aleatoria y queremos publicarla o reproducirla en otros dispositivos, tenemos que generar siempre los mismo números aleatorios, por tanto utilizaremos semillas, de las cuales hablaremos ulteriormente.

GIT

Cuando tenemos un proyecto de gran envergadura y queremos realizar cambios sobre él, hay que ir con especial cautela para operar en el proyecto.

Cualquier cambio en producción que previamente no se ha testeado, cualquier subida a un directorio y puesta inmediata en ejecución sin hacer una copia de seguridad o simplemente no llevar a cabo una política de control de versiones pueden ser problemas muy graves que nos harán perder muchísimo tiempo. Además de que no tendremos un control absoluto y exhaustivo de nuestro código.

¿Cómo controlo las versiones? ¿Y si quiero hacer un experimento nuevo y no tiene éxito? ¿Y si hecho de menos un determinado código de hace 3 meses?

Todo esto afortunadamente tiene solución gracias a unas herramientas denominadas *control de versiones*, que se encargan de guardar todos y cada uno de los cambios importantes que nosotros creamos convenientes en nuestros proyectos.

Uno de los sistemas de control de versión más extendido y utilizado es GIT (7), creado por Linus Torvalds.

Lo que hace especial a GIT sobre otros de su misma especie es la manera con la que tiene de almacenar y tratar los cambios de código.

Representación gráfica de cómo Git almacena los cambios, (fondo verde)
http://image.slidesharecdn.com

Las demás herramientas de control de versiones se encargan de guardar todos y cada uno de los cambios y almacenarlos en una lista. Estos archivos gestionan la información que se ha ido cambiando a lo largo del tiempo de

vida del proyecto. En GIT es algo distinto, cuando nosotros confirmamos algo guarda el estado completo del proyecto, esto no quiere decir que guarde todos los archivos nuevamente, sino que almacena nada más que los cambiados en el proyecto como norma primordial de optimización.

ENTENDIENDO SU ESTRUCTURA Y FUNCIONAMIENTO

Lo básico para poder operar con esta herramienta de control de versiones es saber algo más sobre su composición y funcionamiento. Para ello debemos tener conocimiento de la ubicación de los archivos, de qué es un repositorio y las diferencias que hay entre local y remoto. A continuación se describirán los 2 tipos de entornos que tendremos en nuestro repositorio GIT:

REPOSITORIO LOCAL

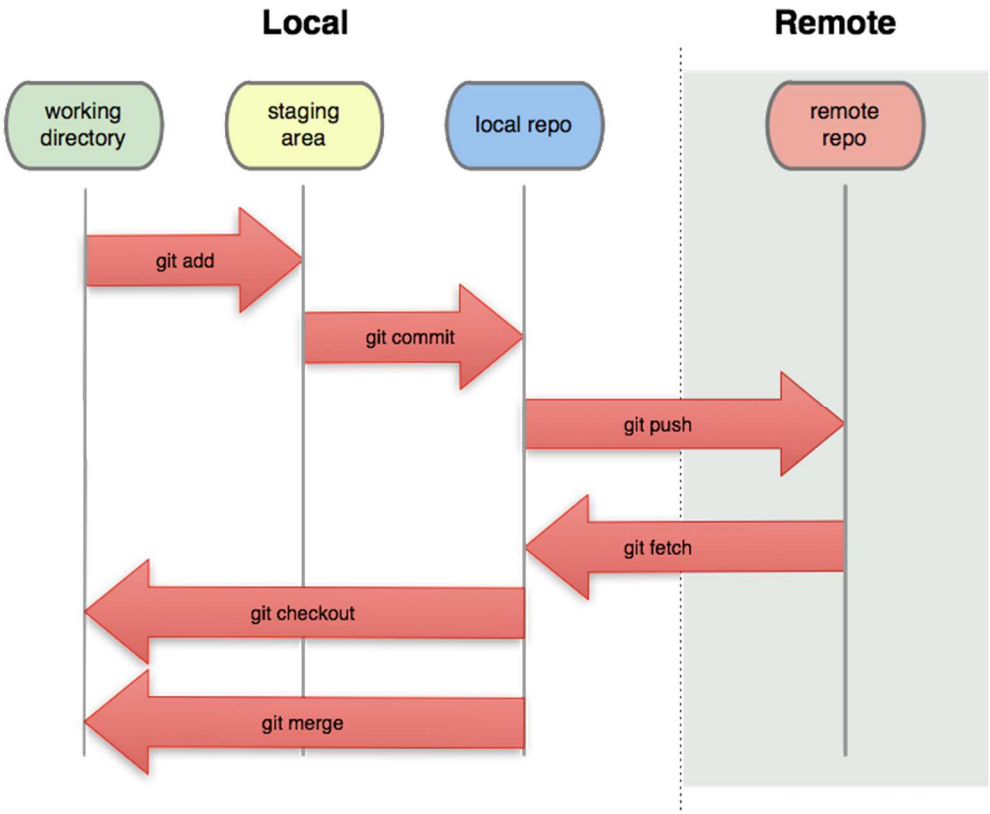
Cuando nos descargamos un proyecto existente de algún servidor en nuestra máquina, automáticamente nos generamos una copia del mismo incluyendo todas las modificaciones desde el primer día de creación del proyecto.

Nosotros podemos modificar los archivos y tenerlos guardados en nuestra máquina para subirlos posteriormente, a parte de modificar también podemos añadir, borrar o realizar cualquier operación de ficheros.

REPOSITORIO REMOTO

Este repositorio es por así decirlo el gestor de git, el repositorio central por el cual todos los usuarios acceden a él. Cuando en nuestro repositorio local hacemos un cambio y queremos propagarlo para el resto de la comunidad, es de vital importancia preparar nuestros cambios y posteriormente hacerselo saber al repositorio remoto. Una vez que tenga los cambios el repositorio remoto, los demás participantes del proyecto podrán efectuar dichos cambios y recibir las modificaciones oportunas.

VINCULACIÓN LOCAL / REMOTO



Relación entre los repositorios locales y remoto

http://michelletorres.mx

Como podemos observar en la ilustración de arriba, la manera de comunicar a ambos repositorios es mediante comandos Git. Son una serie de comandos muy fáciles de usar y que dependiendo de lo que deseemos hacer usaremos una secuencia de ellos o no.

- **Working directory:** este sería nuestro directorio de trabajo, cuando nos descarguemos el proyecto y empecemos a trabajar en él.
- **Staging area:** es la zona donde un proyecto está preparado para ser registrado con un código (*hash*). Este código luego forma parte del repositorio remoto y posteriormente se le informará al remoto de este cambio.
- **Local repo:** repositorio local, nuestro git cliente por decirlo de alguna manera.
- **Remote repo:** repositorio remoto al cual efectuamos futuros cambios.

COMANDOS BÁSICOS USADOS

En el anterior apartado se han comentado la serie de vínculos que tiene un repositorio local con uno remoto y la forma que tienen de comunicarse. Ahora explicaremos los comandos que hacen falta para poder relacionar nuestros cambios con los del repositorio remoto.

Una vez que ya tengamos nuestro servidor remoto localizado deberemos clonarlo en nuestra máquina.

```
git clone [URL] [DIR]
```

Este comando nos clona el proyecto Git ubicado en la URL descrita y lo copia en la carpeta [DIR] dentro del directorio desde donde lo ejecutaremos.

```
git fetch
```

Nos actualiza el proyecto y nos trae del repositorio remoto todas las novedades.

```
git branch –a
```

Lista todas y cada una de las ramas del proyecto. Un proyecto puede tener más de una rama para el desarrollo, normalmente en la creación del mismo hay una rama denominada ***master*** que es la generada por defecto.

```
git add –-all
```

Añadimos los nuevos ficheros que no tenía nuestro git bajo control.

```
git commit [-m Mensaje]
```

Es el encargado de preparar nuestro nuevo trabajo y convertirlo en un nodo más del árbol git. Siempre lleva un mensaje informativo indicando los cambios realizados en el proyecto.

```
git push [repositorio] [rama]
```

Cuando tenemos todos los cambios hechos y hemos realizado el commit de los mismos, procederemos a realizar el push a una determinada rama de un repositorio si lo estimamos oportuno.

```
git pull [repositorio] [rama]
```

Con este comando nos cercioramos de que nos estamos trayendo todas las actualizaciones de una determinada rama dentro de un repositorio dado.

GitHub

GitHub (8) es una plataforma vía Web que ofrece un servicio de compartición de software, en esta plataforma se almacenan proyectos utilizando el sistema de control de versiones denominado anteriormente, Git.

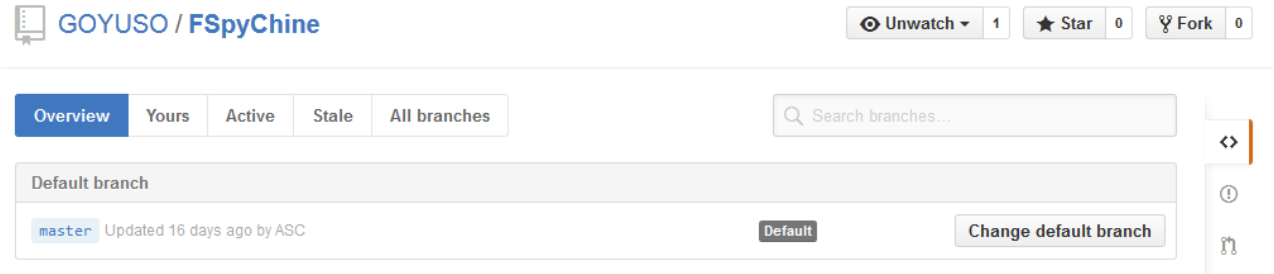
Como nuestro proyecto lo queremos compartir de una forma totalmente libre y gratuita, no tenemos ningún tipo de incoveniente en almacenarlo aquí en esta plataforma, ya que todos los proyectos que residen en GitHub se almacenan de manera pública para que los vea cualquier persona.

NOTA: También existe la posibilidad de hacerlo privado, pero tiene un coste.

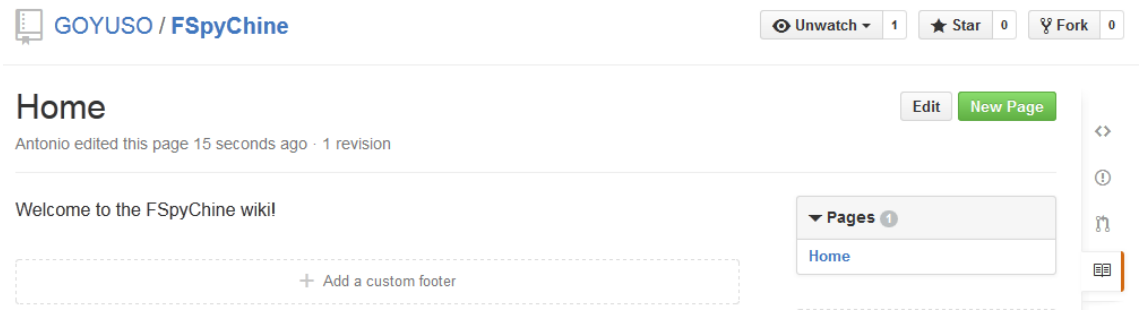
GitHub, además de ofrecerte un servicio gratuito pone a tu disposición una serie de herramientas muy útiles para el trabajo en comunidad.

Algunas de las herramientas a destacar son:

- Visor de ramas.
  Esta funcionalidad básica de GitHub nos permite gestionar y visualizar notablemente todas y cada una de las ramas por las cuales se divide un determinado proyecto. En nuestro caso nada más tenemos una rama remota denominada ***master***.

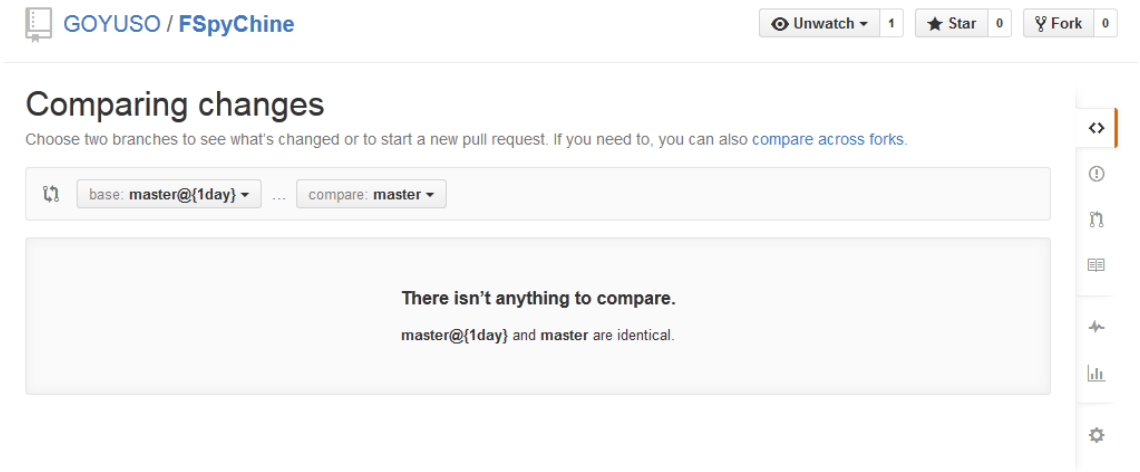


- Wiki.

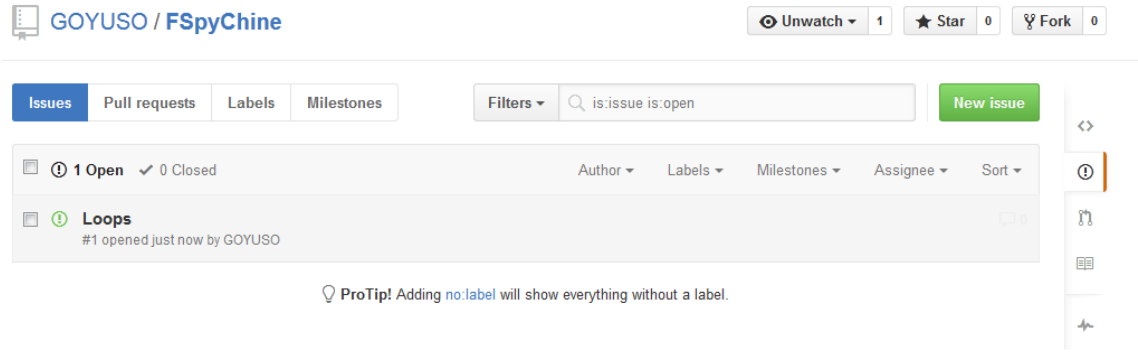


Por otra parte tenemos la Wiki, esta sección adicional de esta plataforma nos da soporte para publicar errores de la herramienta y cómo solucionarlos, instalación paso a paso o cualquier otra instrucción que nos sirva de ayuda.

- Herramienta de revisión de código.
Cuando un proyecto se hace grande o simplemente consigue mucha reputación, creará una serie de colaboradores que lleven a cabo acciones de reforma, entre ellas la de mejora de código y archivos. Para hacer esta tarea más amena se hacen los ***pull request***, que son aportaciones de los colaboradores con los respectivos cambios y mejoras, para que el desarrollador o desarrolladores principales den el visto bueno y lo publiquen.

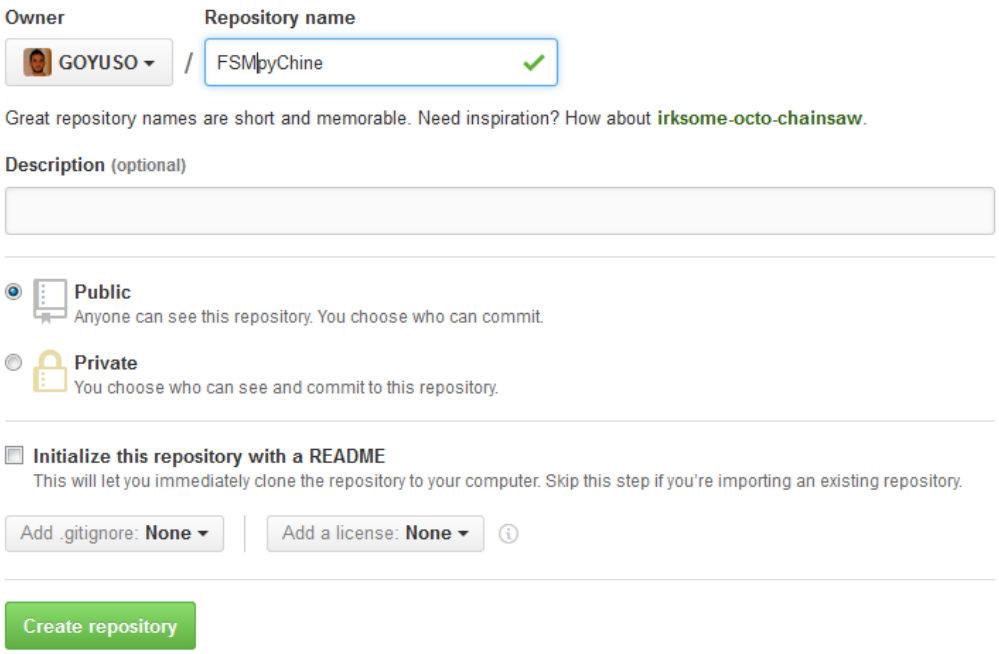


- Apartado de problemas (issues)
  No hace falta ser desarrollador para poder reportar cualquier fallo o anormalidad del proyecto. Para ello GitHub tiene a nuestra disposición el apartado de ***issues***.



Creación de repositorio en GitHub

Con nuestra cuenta GitHub procedemos a crearnos un nuevo repositorio, sobre el cual nos pedirá dos datos obligatorios: el nombre y el tipo de repositorio que queremos (publico o privado).



El resto es usar git en nuestro cliente, clonar el proyecto y utilizar nuestro servidor Git.

Lenguaje Python

INTRODUCCIÓN

En este apartado se enseñará lo imprescindible y básico del lenguaje Python (9) junto con todas las directivas que salen en el proyecto.

Este lenguaje recibe su nombre del programa de televisión británica de la BBC “Monty Python’s Flying Circus”, aunque algunas personas atribuyen erroneamente este nombre con el del reptil. Python fue creado a finales de los ochenta, aunque no saldría a la luz hasta el año 1991.

Python es un lenguaje de programación de alto nivel, fácil de aprender y con una sintaxis muy poderosa.Cuenta con una estructura de datos óptima y una programación orientada a objetos a la par que declarativa.

Además aprovechando de que es software libre, podemos implementar nuevos tipos de variables en C o C++.

Python es un lenguaje interpretado que permite escribir programas compactos y legibles. AdemÃ¡s son generalmente más cortos en líneas de código que sus equivalentes en otro lenguaje, como por ejemplo Java o C#.

Otras de las caracteristicas de Python es que se obvia el uso de llaves de apertura y cierre, en sustitución a estas, usaremos identación o sangría. Por otra parte no es un lenguaje fuertemente tipado, esto quiere decir que no hace falta o no es necesario declarar variables ni argumentos con un tipo específico.

Python es extensible, como se mencionó antes, si queremos hacer eficiente una parte de nuestro código podemos programarla en C e importar ese módulo.

TIPOS DE DATOS

Enteros y flotantes

Son utilizados para representar valores numéricos, los enteros no poseen decimales, luego suelen ocupar también menos memoria. Y luego tenemos los flotantes, que contienen decimales. Las operaciones con números en coma flotante son generalmente y en la mayoría de los casos más lentas que las operaciones con enteros.

Uno de los fallos más comunes a la hora de programar y hacer operaciones con números es la de hacer una división entre dos números enteros tales como el 3 y el 4. Si nosotros abrimos un intérprete de Python y escribimos 3/4 no nos saldrá 0.75, sino 0. Este fenómeno ocurre puesto que python reconoce tanto al 3 y al 4 como números enteros, y su operación entre ellos también la convierte en un entero, truncando el resultado de la operación. (0.75 → 0)

Para evitar este problema, se debe de convertir al menos uno de los dos números a coma flotante: 3.0/4.

Valores lógicos

De valores lógicos sólo hay dos tipos: verdadero o falso. Se pueden llamar también booleanos. Python ofrece soporte para estas operaciones con los operadores lógicos ***True*** y ***False***.

Los operadores lógicos son tres: ***and, or*** y ***not***.

Y sus operaciones son las siguientes:

and		
operandos		resultado
izquierdo	derecho	
True	*True*	*True*
True	*False*	*False*
False	*True*	*False*
False	*False*	*False*

or		
operandos		resultado
izquierdo	derecho	
True	*True*	*True*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

not	
operando	resultado
True	*False*
False	*True*

También se mostrarán las distintas asociaciones con las variables y el orden de prioridad que poseen.

operador	comparación
==	es igual que
!=	es distinto de
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4
Igual que	==	Binario	—	5
Distinto de	!=	Binario	—	5
Menor que	<	Binario	—	5
Menor o igual que	<=	Binario	—	5
Mayor que	>	Binario	—	5
Mayor o Igual que	>=	Binario	—	5
Negación	not	Unario	—	6
Conjunción	and	Binario	Por la izquierda	7
Disyunción	or	Binario	Por la izquierda	8

Tipo de datos cadena

Hasta ahora hemos visto que Python es capaz de operar sin problemas con números enteros y decimales, con valores lógicos y que es capaz de hacer operaciones un poco complejas.

Python también puede manipular los datos de cadena, ¿qué es una cadena? Pues una cadena es una secuencia de letras, números, espacios, marcas que se distinguen por ir cerradas entre comillas simples o dobles.

Tipo de datos lista

Es por así decirlo un array de valores que van en fila, no tienen porqué ser del mismo tipo, ya que nos podemos encontrar estructuras tales como:

[1,"Hola","mundo",2.3]

Tipo de datos diccionario

Es uno de los tipos de datos más útiles en Python. Se considera como un array asociativo o un map de Java. Es mejor pensar de que un diccionario es un conjunto desordenado de parejas clave valor.

FUNCIONES

Funciones predefinidas

Hemos estudiado los operadores aritméticos básicos. Python también proporciona funciones que podemos utilizar en las expresiones. Estas funciones son predefinidas por el propio lenguaje. Algunas de las que se han usado en el proyecto son:

int: conversión a entero. Si recibe un número flotante como argumento, devuelve el entero que se obtiene eliminando los decimales.

str: conversión a cadena. Recibe un número y lo devuelve expresado como cadena.

Funciones definidas en módulos

Hay dos maneras de importar una función dado un módulo, y en muy pocos lugares explican la diferencia. La primera declaración se hace así:

```
from FSM_utils import log
```

de esta manera sólo importa la función log, que es la que deseamos. Claro está que si queremos importar todas las funciones podemos escribir lo siguiente:

```
from FSM_utils import *
```

Aunque no resulta aconsejable por varias razones, la primera es que el programa pierde legibilidad pues no sabemos de donde vienen las funciones. Y segundo, si hemos definido una variable que coincide con el nombre de una función definida en el módulo, nuestra variable será sustituida por la función.

Como en este trabajo el módulo que usamos es casero y posee un número limitado de funciones, procederemos a importar todo el módulo.

La segunda manera de realizar una importación es directamente escribiendo:

```
import FSM_utils
```

De esta forma, todas las funciones del módulo estarán disponibles, pero usando el nombre del módulo como prefijo.

Métodos

Dependiendo del tipo, algunos permiten invocar unas funciones especiales: los denominados métodos. Un método nos permite hacer una operación con dicho tipo, como por ejemplo pasar una cadena a mayúsculas.

ESTRUCTURAS DE CONTROL

Sentencia condicional IF

La sentencia IF en Python tiene la siguiente estructura:

```
if condición:
    acción
    acción
```

La condición siempre tiene que ser un valor que nos identifique si es verdadero o falso. Es recomendable usar un valor booleano, pero también se pueden utilizar números, siendo 0 el falso y cualquier otro el valor verdadero.

Puede haber cero o más bloques ***elif***, y el bloque ***else*** es opcional.

Sentencia iterativa WHILE

```
while condición:
    acción
    acción
    ...
    acción
```

Para usar este tipo de sentencia, hay que tener en cuenta que vamos a realizar una serie de instrucciones o acciones de forma iterativa hasta que la condición deje de cumplirse.

Las sentencias que denotan repetición se denominan bucles.

Sentencia iterativa FOR IN

```
for i in range(valor inicial, valor final + 1):
    acciones
```

En lugar de siempre iterar sobre una progresión aritmética de números o darle al usuario la posibilidad de definir tanto el paso de la iteración como la condición de fin, la sentencia ***for*** de Python itera sobre los ítems de cualquier secuencia, ya sea cadena o lista, y la recorre en el orden estipulado.

Tratamiento de errores

```
try:
    acción potencialmente errónea
    acción potencialmente errónea
    ...
    acción potencialmente errónea
except:
    acción para tratar el error
    acción para tratar el error
    ...
    acción para tratar el error
```

Cuando tenemos un fragmento de nuestro código que no podemos evitar tener errores ya que dependen de otros factores que no podemos abarcar nosotros (por ejemplo, en nuestro proyecto la correcta introducción de los parámetros a la hora de instanciar una clase), tenemos que utilizar una estructura de trata de errores, y es la ilustrada arriba. En la sección ***try*** introducimos las intrucciones con errores potenciales y en la excepción el tratamiento de los mismos en el caso de que haya algún error.

Esta estructura es muy útil, puesto que al haber un error, el programa no para automáticamente, simplemente te avisa o lo camufla y prosigue con su ejecución.

PIP

Pip (10) es un sistema que gestiona los recursos o los paquetes de Python y sus siglas provienen de un acrónimo recursivo (***Pip Installs Packages/Python***) Pip Instala Paquetes/Python.

Lo que más favorece el uso de pip es la facilidad de uso que éste conlleva a la hora de desempeñar las funcionalidades que nosotros deseamos y que, desde un principio, nos pueden resultar un tanto tediosas debido al complejo mecanismo o forma de instalar los paquetes que puede tener un determinado paquete de Python. Esta ventaja se consigue mediante el CLI (***Command Line Interface***) Interfaz de línea de comandos con la siguiente estructura:

```
pip install nombre_del_paquete
```

Y si queremos desinstalar un paquete la sintaxis es la misma:

```
pip uninstall nombre_del_paquete
```

Instalación

A partir de la versión de Python 2.7.9 no es necesario instalar pip, pero para los desarrolladores que tengan una versión más obsoleta pueden hacerlo de la siguiente manera como administrador:

En Windows:

```
python get-pip.py
```

En Linux y Mac OS:

```
sudo python get-pip.py
```

Actualización

Si queremos actualizar pip tendremos que ejecutar el siguiente comando:

En Windows:

```
python -m pip install -U pip
```

En Linux y Mac OS:

```
pip install -U zip
```

PEP 8

Introducción

Pep 8 (11) es una guía de prácticas que nos brinda un estilo para la mejora de la legilibilidad del código, y hacerlo por lo tanto lo más consistente posible dentro de la comunidad Python. Esta guía fue escrita por Guido van Rossum y Barry Warsaw, y a continuación abarcaremos algunos aspectos que han sido tomados en cuenta durante la realización del proyecto.

Diseño del código

Para la identación se usarán cuatro espacios exclusivamente, antiguamente se solía usar una identación doble, es decir, de ocho espacios. En este caso sí se puede continuar usando esas identaciones para no estropear el código.

Las líneas de continuación siempre deben alinearse aplicando tabulaciones en todas las líneas con excepción de la primera o bien alinearlas verticalmente con el carácter que se ha utilizado.

```
# Alineado con el paréntesis que abre la función
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Más indentación para distinguirla del resto de las líneas
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

Tabulaciones y espacios

Bajo ningún concepto han de mezclarse ambos. Se pueden usar ambos, o bien espacios, o bien tabulaciones, pero nunca juntos. De todas formas, cualquier código identado con una mezcla de ambos, debe de ser convertido obligatoriamente a espacios.

Para saber si nuestro código contiene una mezcla de los dos, siempre es conveniente arrancar la consola de Python con la opción `-t` o `-tt`. La primera opción muestra con un mensaje de advertencia si un código mezcla tabulaciones y espacios, la segunda opción convierte la advertencia en un error.

Si utilizamos un IDE o editor de texto de alto nivel, no nos tendremos que preocupar demasiado por la identación, puesto que en la mayoría de los casos la hace automáticamente.

Longitud de líneas

Las líneas se limitan todas a un máximo de 79 caracteres, a excepción de los largos bloques de documentación denominados ***DocStrings*** o comentarios, los cuales también pueden ser limitados a 79 pero es recomendable ponerlos bajo el umbral de los 72 caracteres.

Líneas en blanco

Las funciones y las definiciones de clase se deben de separar con dos líneas en blanco, las definiciones de métodos en cambio deben de ser separadas únicamente con una sola línea en blanco.

Importaciones

Las importaciones deberán estar siempre en líneas separadas a excepción de cuando queremos importar de un módulo unas cuantas funciones.

```
from modulo import Funcion1, Funcion2
```

Las importaciones deberán ir siempre al principio del archivo y siempre en el siguiente orden separadas en grupos por una línea en blanco:

1. Importaciones de bibliotecas estándar.
2. Importaciones de bibliotecas terceras.
3. Importaciones locales de nuestra aplicación.

Expresiones y sentencias

Evitar el uso de los espacios en blanco inmediatamente dentro de los paréntesis, corchetes o llaves:

`Funcion(param1, param2)` y no `Funcion( param1, param2 )`

También es aplicable al uso de las comas, punto y coma o dos puntos.

No está permitido alinear con un operador de asignación las demás variables, sólo hay que dejar un espacio.

Argumentos de la clase

En la creación de la clase, el primer argumento del método constructor es nombrado ***self***, mientras que el primer argumeto de los métodos de esa determinada clase siempre usaremos ***cls***.

Variable privadas

Una variable privada nos va a ser de real ayuda si tenemos pensamientos de incorporar nuestro módulo a otros proyectos, como puede haber un alto riesgo de que coincidan variables de este módulo con algunas ya creadas, se hacen esas variables privadas. Para reconocer una variable privada, tan solo hay que añadir dos guiones bajos `__`.

TKINTER

Para realizar la interfaz gráfica se ha utilizado Tkinter (12), Tkinter es una biblioteca muy útil para hacer interfaces guiadas para usuario (GUI) que pertenece a Python. Esta pequeña aplicación nos proporciona un programa a nivel visual en cualquier plataforma, ya que es compatible tanto en sistemas UNIX como en Windows.

Para desarrollar la interfaz de la aplicación del proyecto se han necesitado los siguientes conocimientos de esta herramienta.

Definiciones

- Window: área de tipo rectangular en la cual va embutido todo nuestro contenido de la aplicación.
- Widget: cualquier objeto representado dentro de nuestra ventana, por ejemplo los botones, radios, campos de texto y etiquetas.
- Frame: es un tipo de widget que representa la organización de las capas de diseño de nuestra aplicación. Un frame es un widget capaz de contener otros widgets.

Posicionamiento o empaquetado de los widgets

El posicionamiento se ha representado con el método ***grid***, cada widget posee este método. Sea ***wg*** un objeto instanciado de tipo widget, se ubicará de la siguiente manera:

OPCIÓN	Explicación
column	El número de columna de la matriz visual donde queremos ubicar nuestro objeto. Defecto: 0
row	El número de la fila de nuestra matriz visual donde deseamos ubicar nuestro widget. Defecto: el siguiente número que no tenga ocupando nada otro widget.
sticky	Es un espacio extra. Es representado con las siglas de posicionamiento de una brújula (N,S,E,W,NE.SE,SW,NW)

Widgets

A continuación se describirán los widgets utilizados en el proyecto. Son los siguientes:

Botón

Este widget es uno de los más utilizados y conocidos, su funcionamiento es trivial, declaramos un objeto, le asignamos el frame al que queremos embutirlo y le añadimos una serie de parámetros a nuestro gusto.

Entre todos los parámetros y múltiples opciones que nos ofrece este widget, hemos usado en el proyecto estos:

OPCIÓN	Explicación
text	Texto que queremos que aparezca en nuestro botón una vez creado.
command	Una vez pulsado el botón, llama al evento de click y a la función que tengamos definida dentro.

Ejemplo:

```
def exportKiss2():
    method_name = "fsm_" + str(self.v.get())
    method = getattr(self, method_name)
    return method()

b1 = Button(master, text="Export kiss2", command=exportKiss2)
# b1.pack()
b1.pack()
```

Label

Este objeto lo único que nos proporcionará en nuestro programa es otorgar información sobre los campos a rellenar.

Los parámetros de instanciación son los siguientes: frame y opciones.

OPCIÓN	Explicación
Text	Campo de tipo cadena que muestra en la interfaz lo que queramos.
bg	Color de fondo que deseamos atribuirle, en la aplicación se usa el blanco (#ffffff)

Entry

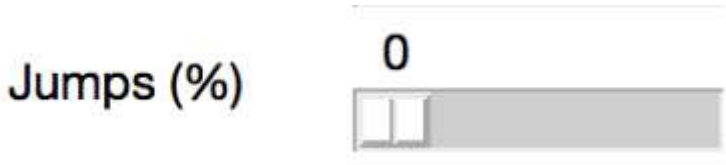
En el proyecto, este objeto va fuertemente vinculado con el objeto ***Label***.

Los parámetros de inicialización son los mismos que para Entry:

OPCIÓN	Explicación
Textvariable	Es el valor que queremos mostrar como variable dentro del campo de texto.

Scale

Este tipo de widget es el encargado de proporcionarnos las dos barras deslizantes que otorgan valores del 0 al 100 (%).



OPCIÓN	Explicación
Orient	Indica la orientación de la barra deslizadora, siendo HORIZONTAL o VERTICAL sus posibles valores. Defecto: VERTICAL.
Sliderlenght	Longitud física de la barra dentro del marco de la aplicación.
Variable	Valor de lectura y escritura durante la vida del programa
Bg	Nos indica el color de fondo que queremos para dicha barra

Radiobutton

El último objeto gráfico que representamos en nuestra interfaz es el de los radios.

OPCIÓN	Explicación
Text	Es la cadena que acompaña a la opción del radio
Variable	Valor de lectura y escritura durante la vida del programa
Value	Valor que tomará la variable asignada al conjunto de radios si se selecciona su opción
Command	Función que se ejecuta cuando pulsamos en una opción de tipo radio
Bg	Nos indica el color de fondo que queremos para dicha barra

Automatización de nuestros widgets

Aquí un ejemplo de automatización de todos los objetos de representación gráfica.

```
# inputs = [
#     ("Min number of bits", "vmin", Entry, 1),
#     ("Max number of bits", "vmax", Entry, 3),
#     ("Seed", "seed", Entry, "mySeed"),
#     ("Number of states", "states", Entry, 5),
#     ("Indeterminacy", "ind", Scale, 0.0),
#     ("Loops", "loops", Scale, 0.0)
# ]
```

```
radios = [
    ("Random", 1),
    ("Sequential", 2),
    ("Patterns", 3)
]
self.v = IntVar()
self.v.set(1)
n=2
for text, value in radios:
    b = Radiobutton(self.frame, text=text, variable=self.v,
    b.grid(row=n, column=0, sticky=W)
    n += 1
```

```
for text, value, f, default in inputs:
    Label(frame, text=text, bg=self.MAIN_COLOR).grid(row=n,column=3,sticky=E)
    if f == Entry:
        if type(default) is int:
            self.results[value] = IntVar()
        else:
            self.results[value] = StringVar()
        t1 = f(frame, textvariable=self.results[value])
        t1.grid(row=n,column=4)
    if f == Scale:
        self.results[value] = IntVar()
        f(frame, orient=HORIZONTAL, sliderlength=20,variable=self.results[value],

    self.results[value].set(default)
    n += 1
```

Especifiones

Las especificaciones de la máquina sobre la cual se ha realizado este proyecto son las siguientes:

Procesador: Intel Core i7 de cuatro núcleos a 2,2 Ghz con 6MB de caché (Nivel 3) compartida.

Memoria: 16 GB de memoria DDR3L integrada a 1600 MHz.

Almacenamiento: 256 GB de almacenamiento flash PCIe.

Sistema Operativo: Mac OS X Yosemite

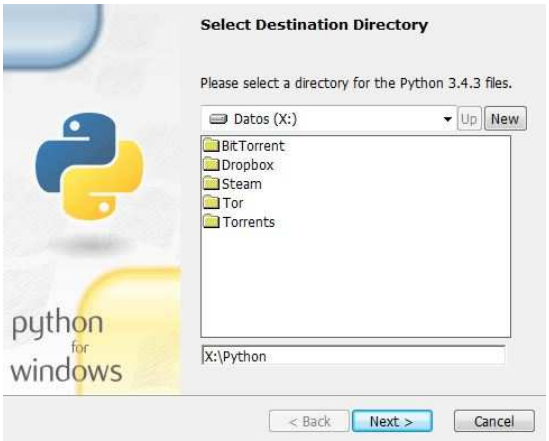
Instalación

Para poder ejecutar satisfactoriamente el proyecto tendremos que seguir los siguientes pasos:

INSTALANDO PYTHON2

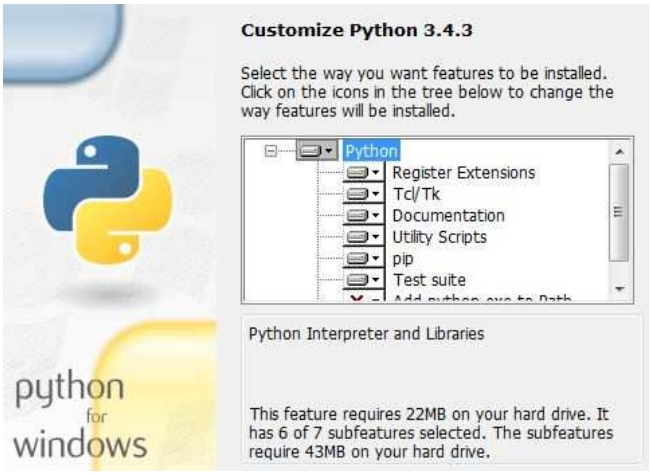
Si no tenemos Python instalado en nuestra máquina, tendremos que seguir unos sencillos pasos.

Deberemos de ir a la sección de descargas de la página oficial de Python (https://www.python.org/downloads/) dependiendo de nuestro sistema operativo y necesidades elegimos nuestro instalador.



Una vez que descargamos nuestro instalador, lo ejecutamos y nos mostrará un asistente preguntándonos dónde deseamos almacenar el intérprete de Python.

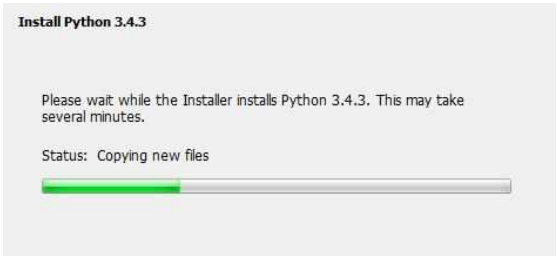
Cuando lo hayamos elegido, pulsamos ***Next*** y saltamos a la siguiente pantalla.



El siguiente paso a realizar es el de la instalación de las herramientas, paquetes o bibliotecas que nos ofrecerán una serie de características adicionales al lenguaje y que nos facilitarán las horas de desarrollo. Entre ellas es recomendable, o más bien, de obligatorio cumplimiento, instalar ***pip*** .

Esta herramienta nos ayudará a instalar los demás paquetes que vamos a necesitar para que nuestro proyecto funcione adecuadamente.

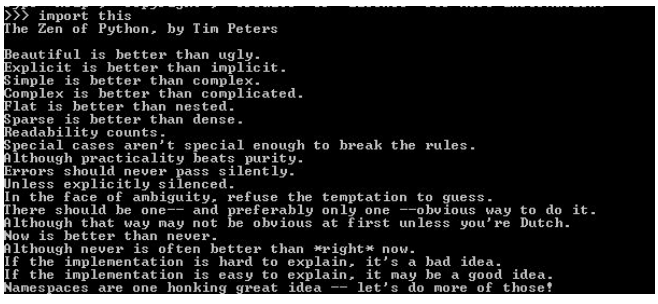
Una vez que hayamos decidido qué instalar en adición y qué no procederemos a su instalación. Esperamos al proceso de instalación mediante una barra de progresos.



Por último, si todo ha salido correctamente, un mensaje de que la instalación se ha efectuado de manera satisfatoria se nos mostrará.

Pulsamos ***Finish*** (Finalizar) y ya tenemos nuestro intérprete de Python listo para nuestras necesidades.

Podemos comprobar que nuestro intérprete funciona perfectamente realizando cualquier simple operación aritmética o cualquier instrucción apta para Python, como por ejemplo:




```
import this
```

Que nos mostrará los mandamientos o principios de Python escritos por Tim Peters.

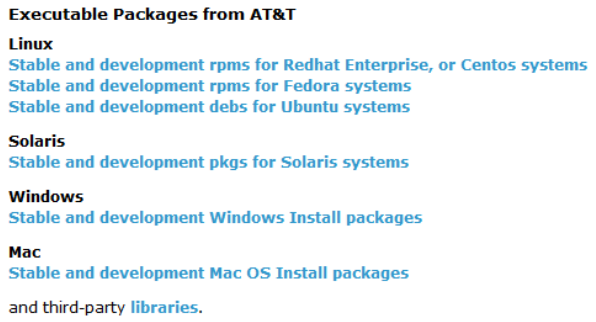
INSTALANDO GRAPHVIZ DOT

Para poder generar la imagen de la máquina de estados y exportarla a nuestra máquina debemos tener está biblioteca instalada.

Para ello tendremos que entrar en la sección de descargas de la página oficial de Graphviz (13) (http://www.graphviz.org/Download.php)

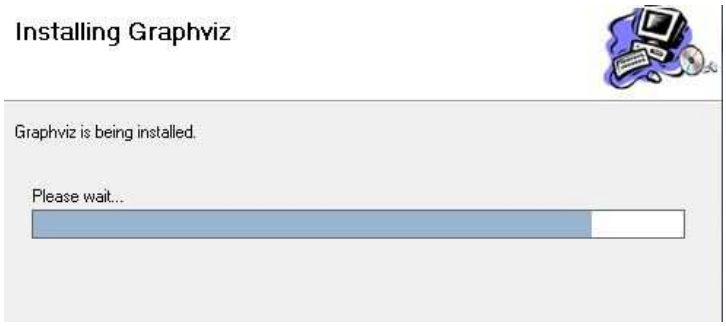
Lo primero que tenemos que hacer es aceptar las condiciones y términos legales de la licencia. Cuando los hayamos aceptado, nos remitirá a una página con los distintos enlaces de descarga dependiendo de nuestro sistema operativo nuevamente.

Escogemos el nuestro y procedemos a su instalación. Se recomienda descargar la versión que ellos recomiendan, una estable, las de desarrollo suelen tener fallos.



Una vez tenemos el instalador, procedemos a su instalación. Como en el caso de Python, viene acompañado de un asistente muy intuitivo que nos preguntará por la ruta en la que queremos almacenar el proyecto GraphViz.

Por último dejamos que se instale el programa por el asistente y ya lo tenemos instalado.



INSTALACIÓN DE GIT (OPCIONAL)

Si sólo queremos usar el programa bajo el perfil de usuario no necesitamos instalar Git como herramienta. Pero si nuestro cometido es el de mejorarla, pues tendremos que descargarla.

Para ello nos vamos a la página de descargas de Git y es cogemos nuestro sistema operativo (https://git-scm.com/downloads).

Si el usuario en cuestión no se desenvuelve muy bien con los comandos de Git, se recomienda que utilice un ayudante gráfico (*GUI Clients*)

Windows viene con un instalador, pero si queremos instalar Git con pocos comandos en Linux o Mac OS, esto son los pasos a seguir.

Linux tan solo necesita una instrucción:

```
sudo apt-get install git
```

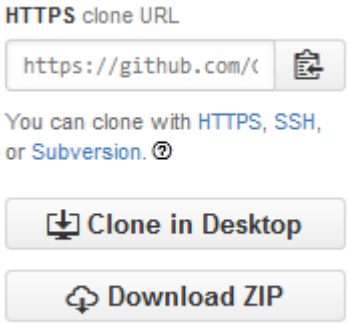
Mac OS. Si no tenemos instalado ***brew*** y queremos hacerlo por línea de comandos, deberemos instalarlo. Brew es el equivalente al gestor de paquetes por consola de Linux *apt-get*, podemos instalarlo de la siguiente manera:

```
ruby -e "$(curl –fsSL
https://raw.githubusercontent.com/Homebrew/install/master/
install)"
```

Una vez que se instala brew, procedemos a instalar git

```
brew install git
```

Anteriormente se ha comentado de que este paso es opcional porque podemos ir directos a la página donde está ubicado el repositorio (https://github.com/GOYUSO/FSpyChine.git) y descargarlo como un .zip. Lo descomprimimos y ya tenemos nuestro proyecto.



Convención de versiones

La metodología por la que nos basaremos para nombrar las distintas versiones de este proyecto desde su inicio y en un futuro serán descritas a continuación.

La versión se compondrá por tres números separados por puntos y el primero prefijado por una *v* minúscula:

`v0.5.1`

El primer número para nuestro proyecto indicará los cambios funcionales drásticos que sufrirá la herramienta. Por ejemplo, si nuestra herramienta pasa de ser por consola a poseer una interfaz gráfica, sumará un punto a la versión ya que hemos hecho un cambio drástico.

El segundo número será destinado al número de cambios estructurales de baja o mediana importancia que acontecerán sobre la herramienta. Por ejemplo, la inserción de un nuevo método o una nueva funcionalidad para la creación de otro tipo de máquina.

Y por último, el tercer apartado es el de los fallos que se han corregido. No necesariamente por cada fallo debe de aumentar en uno su incremento, sino que pueden solucionarse varios fallos o incidencias en una misma versión.

De esta manera la versión descrita como ejemplo nos querrá decir que estamos ante una versión del proyecto inmadura, con 5 cambios funcionales aproximadamente y con una serie de incidencias subsanadas.

¿CUÁNDO PASAR A LA VERSIÓN 1?

La versión 1 es la primera versión del proyecto, suele ser lanzada cuando la consideramos estable, o lo que es lo mismo, está listo para ser lanzado en producción.

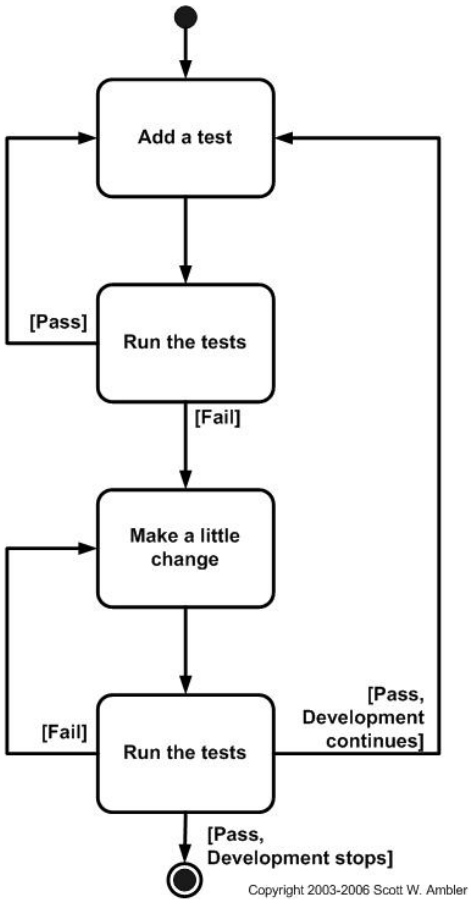
Esta herramienta no tiene la necesidad de estar en un entorno de producción puesto que va a ser utilizada por profesionales, aunque esperaremos a una versión madurada para decidir pasar a la versión v1.0.0.

TDD

TDD (14) proviene de las siglas inglesas *"Test-driven development"* o en español, *"Desarrollo guiado por pruebas"*. Es un enfoque evolutivo para el desarrollo que combina el desarrollo de la prueba primero en el que se describe una prueba antes de escribir todo el código necesario para cumplir con esa prueba y poder refactorizarlo.

¿CUÁL ES EL OBJETIVO PRINCIPAL DE TDD?

Un punto de vista muy útil sobre el cual nos ceñiremos es que el objetivo de TDD es una especificación y no una validación. En otras palabras, es una manera de pensar a través de las necesidades o el diseño antes de escribir su código funcional (lo que implica que TDD es a la vez una importante herramienta para el desarrollo ágil). Otro punto de vista es que TDD es una técnica de programación. El objetivo de TDD es escribir código limpio que funciona.



Formato Kiss2

La herramienta está desarrollada principalmente para exportar máquinas de estados en formato kiss2 (15).

Kiss2 es un formato para representar máquinas de estados finitas. Kiss2 se compone principalmente de dos partes: una cabecera y el cuerpo. En la cabecera se ubican como máximo cinco tipos de datos informativos, que contiene la siguiente información:

	Cabecera Kiss2
.i	Número de bits de entrada que recibe la máquina de estados
.o	Número de bits de salida que se emiten como
.p	Número de productos
.s	Número de estados usados
.r	Estado RESET [Opcional]

Donde tanto la ***i*** com la ***o*** son el número de bits de entrada como de salida que operan en cada uno de los estados. Los productos son todas y cada una de las descripciones que posee la máquina de estados. Los estados usados es un número que nos indica de cuántos estados se compone una máquina. El último componente es opcional, ya que nos da información sobre cual es el estado por defecto (Reset) de la máquina.

NOTA: El número de productos máximo se ve expuesto ante la siguiente fórmula matemática.

$$P \leq 2^i * s$$

Siendo **P** el número de productos, **i** el número de bits de entrada y **s** el número total de estados de la máquina.

Y por otra parte nos queda el cuerpo del formato kiss2, que se rige por la siguiente estructura:

```
<entrada> <estado_actual> <siguiente_estado> <salida>
```

La entrada será una secuencia compuesta por el conjunto {0,1,-} de tamaño ***i***.

Tanto el estado actual como el siguiente estado serán representados con el nombre que le otorguemos en el fichero.

La salida será una secuencia compuesta también por un conjunto {0,1,-} pero de tamaño ***o***.

Clase FSM

La estructura principal por la cual se fomenta el proyecto es mediante la clase que genera las máquinas FSM.

Esta clase es la encargada de instanciar los objetos y trabajar con ellos.

Estructura de objetos

Los objetos generados por la clase FSM tienen la siguiente estructura para ser representados. Siendo X un objeto generado por nuestra herramienta con la clase FSM obtendríamos lo siguiente:

X:

Lista de Paramétros de entrada(entradas, salidas, semilla, opción de bucle...)

Diccionario de máquinas de estados para cada estado hasta N:

Siendo una tupla Tx : (estado_siguiente, valorDeTransición, valorEnSiguienteEstado)

{

“s0”: [T1,T2,...,Tn],

“s1”: [T1,T2,...,Tn],

...

“sN”: [T1, T2,...,Tn]

}

Funciones

En este apartado se hablarán de los tres tipos de funciones que implementa la herramienta FSMpyChine.

Los parámetros que hacen de factor común en nuestra herramienta y que sirven para todas y cada uno de los tipos de máquinas de estado son los siguientes:

Parámetro	Explicación	Tipo
Input	Número de bits de entrada en la máquina de estados	Entero
Output	Número de bits de salida en la máquina de estados	Entero
Número de estados	Cantidad de estados que va a poseer nuestra máquina	Entero
Semilla	Cadena de texto para poder generar nuestra máquina aleatoria en cualquier máquina y poder reproducirla con total fidelidad.	Cadena (String)

RANDOM

A la hora de crear una máquina de estados de tipo random, el usuario podrá crear una máquina de estados con número de entradas y salidas específico, así también como una cantidad de estados predefinida. Se incorpora un cuarto parámetro que es la semilla, este valor de tipo cadena se usará para poder reproducir la misma máquina aleatoria en otra máquina con la total certeza de que va a ser la misma una y otra vez.

Pseudo-código

Procedimiento random(param):

DiccionarioDeEstados ← {}

Np ← creación de la semilla aleatoria

Nstates ← states

listaEstados ← lista de s0 hasta sNstates

Para cada Estado en listaEstados Hacer:

    Para cada Arista en un rango de 2^input:

        Generar tupla : (EstadoDeTransición, V. Entrada, V.Salida)

        DiccionarioDeEstados ← DiccionarioDeEstados + tupla

Devolver diccionarioDeEstados

PATRONES

La máquina de estado por patrones es un tipo de máquina de estados que devuelve una serie de patrones que reconoce. Estos los imprime una vez creada la máquina de estados bajo petición del usuario. Dependiendo de donde se ubique se le mostrará o bien por consola o mediante una ventana en la interfaz gráfica.

Pseudo-código

Procedimiento pattern(param):

DiccionarioDeEstados ← {}

Np ← creación de la semilla aleatoria

Nstates ← states

Output ← param.output

Input ← param.input

listaEstados ← lista de s0 hasta sNstates

Para cada Estado en listaEstados Hacer:

    Para cada Arista en un rango de 2^input:

        V.Entrada ← número en binario de output bits.

        Generar tupla : (EstadoDeTransición, V. Entrada, 0 | 1)

        DiccionarioDeEstados ← DiccionarioDeEstados + tupla

Devolver diccionarioDeEstados

SECUENCIAL

En este tipo de máquina podremos generar una secuencia de estados con un porcentaje asignado de ciclos y saltos. Siendo la suma de estos 2 parámetros menor de 100.

Parámetro	Explicación	Tipo
Loops (ciclos)	Porcentaje de ciclos en un mismo estado	Número [0-100]
Jumps (saltos)	Porcentaje de saltos entre estado y estado.	Número [0-100]

Pseudo-código

```
Procedimiento sequential(param):

Si 100 <= param.Loops + param.Jumps entonces:

        Devolver error

Sino:

        diccionarioDeEstados ← {}

        listaEstados ← lista de s0 hasta param.states

        Para cada Estado en listaEstados Hacer:

                Para cada Arista en un rango de 2^input:

                        V.Entrada ← número en binario de param.output bits.

                        V.Salida ← número en binario de param.input bits.

                        GenerarDado ← mismoEstado | siguiente | otro

                        EstadoTransicion ← listaEstados[generarDado]

                        Generar tupla(EstadoTransición, V. Entrada, V.Salida )

                        DiccionarioDeEstados ← DiccionarioDeEstados + tupla

Devolver diccionarioDeEstados
```

Python DOC – FSM

(CLASS) FSM	Nombre de la clase

def init(**kwargs)	Fúnción de instanciación para los objetos de la clase FSM. Recibe como parámetros todos los atributos que deseemos para crear una máquina de estados.
Ejemplo	X=FSM(states=5, input=3,output=1,loops=40,jumps=20)
Devuelve	Objeto FSM instanciado.

def build(funcion)	Una vez instanciado un objeto (en nuestro caso le hemos llamado X) procederemos a construir nuestro tipo de máquina de estados, habiendo 3 tipos de momento. (random, sequential, pattern) Estas funciones devuelven siempre un diccionario con la estructura descrita anteriormente.
Ejemplo	X.build(sequential)
Devuelve	Nada (tan sólo construye la FSM)

def kiss2([path])	Con este método crearemos nuestro fichero en formato kiss2. Recibe un único parámetro opcional indicando donde se desea guardar dicho fichero. Si no se dice nada se ubicará en la carpeta donde se ha ejecutado el programa.
Ejemplo	X.kiss2("./misFSM/kiss2/sequencial.kiss2")
Devuelve	Fichero kiss2

def image([path])	Método parecido a kiss2 pero que genera una imagen en la ruta indicada. Si no se dice nada se ubicará en la carpeta donde se ha ejecutado el programa.
Ejemplo	X.kiss2("./misFSM/images/sequencial.png")
Devuelve	Imagen PNG

def getPatterns()	Método sólo permitido para los objetos construidos con la función patterns
Ejemplo	X.getPatterns()
Devuelve	Patrones de la FSM (print o mensaje de alerta)

Cómo ejecutar el proyecto

Una vez que ya hemos instalado todas las dependencias: Python2, Graphviz dot, git (muy recomendable)... Procederemos a la ejecución del proyecto.

CON GIT

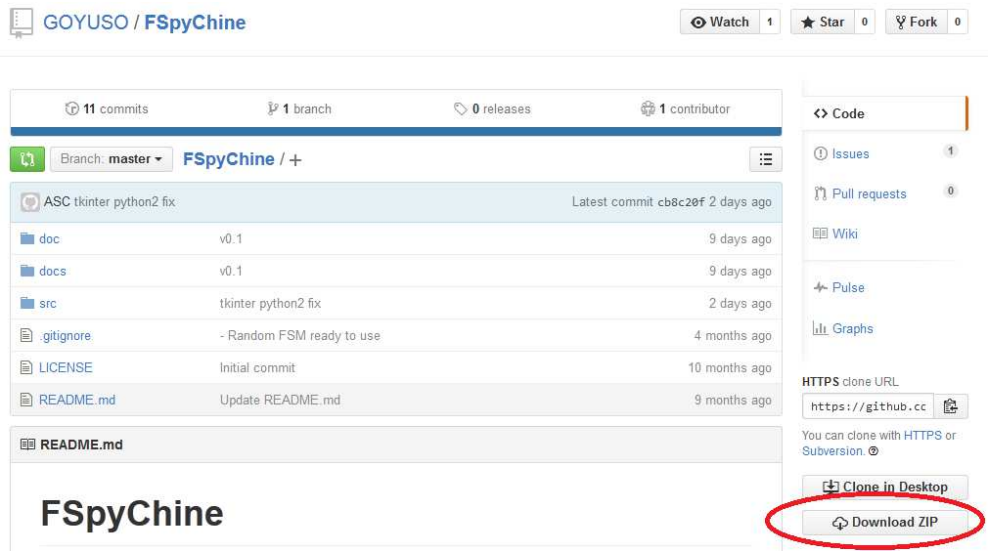
Con git tendríamos que hacer lo siguiente:

Clonamos repositorio:

```
git clone https://github.com/GOYUSO/FSpyChine.git
```

SIN GIT

Nos vamos a la página https://github.com/GOYUSO/FSpychine y le damos a la opción **Download ZIP**



Luego ejecutamos con python nuestra herramienta, dependiendo de la plataforma se hará de una determinada manera u otra.

Sistemas UNIX:

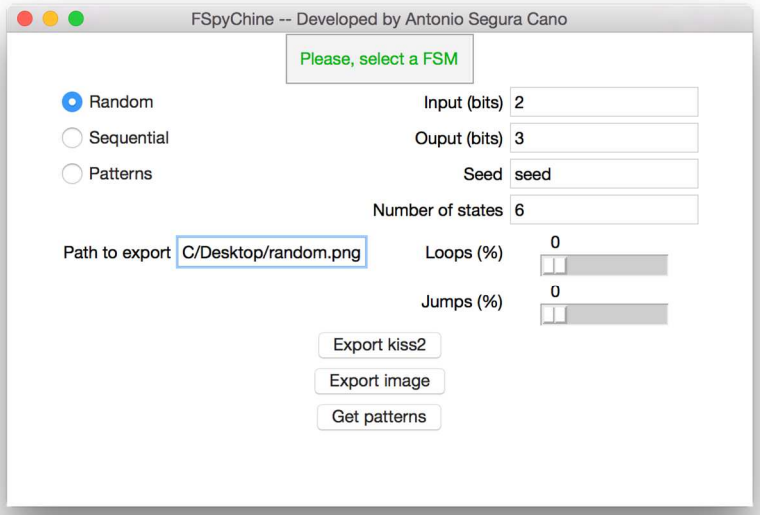
```
cd FspyChine/src && python FSM_gui.py
```

Sistemas Windows:

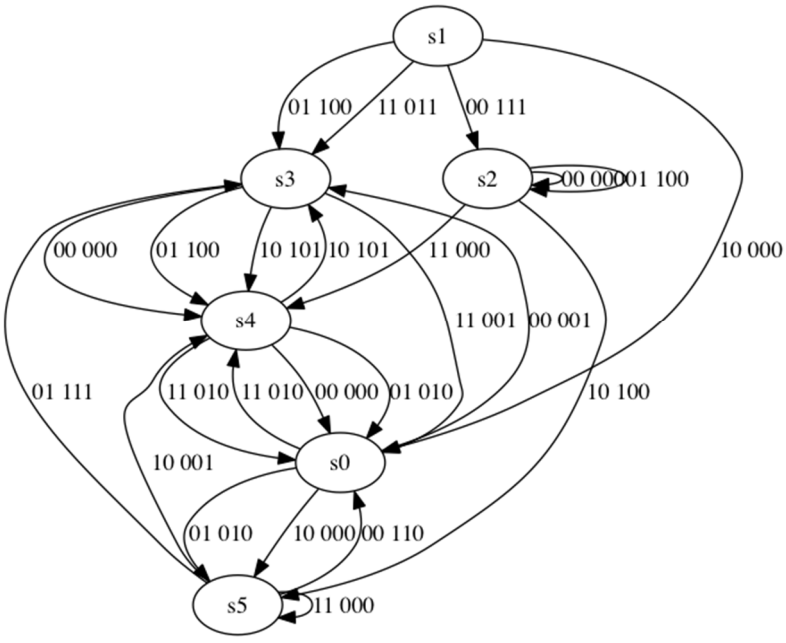
```
X:\RutaPython\python.exe FSM_gui.py
```

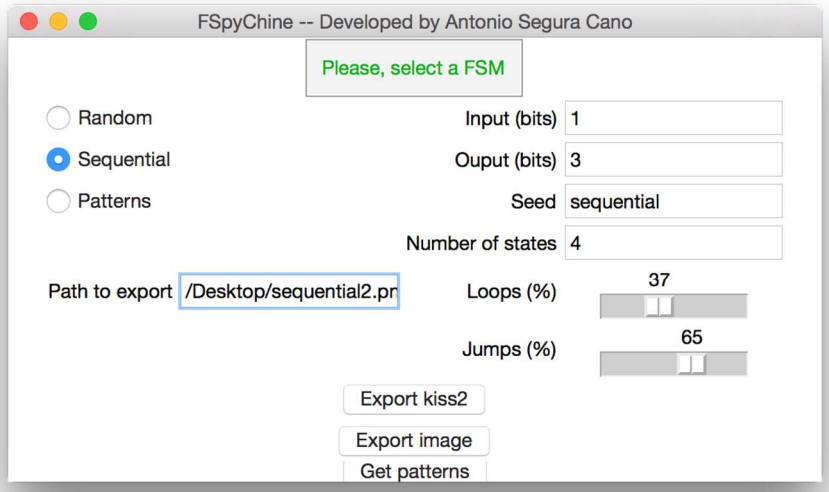
Ejemplos de ejecución

Una vez abierta la aplicación nos saldrá la siguiente interfaz. En la parte izquierda tenemos 3 posibles opciones para elegir nuestro tipo de máquina: aleatoria, secuencial y por patrones.

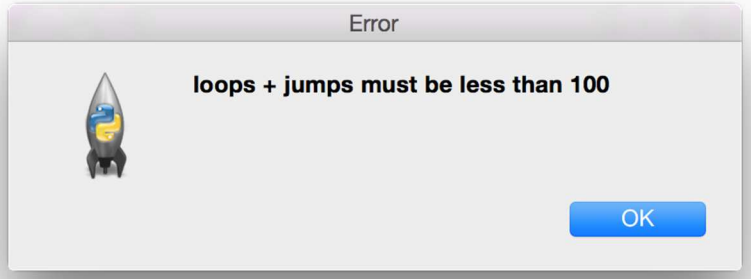


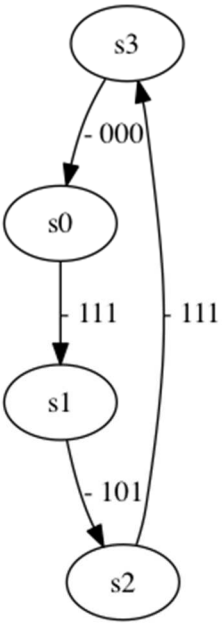
A la derecha tendremos los distintos parámetros a introducir en nuestra máquina de estados. Por último, debajo de la interfaz, poseemos los botones de exportación, estando la variable de ruta de exportación arriba a la izquierda. Con los parámetros de arriba, si pulsamos en ***Export kiss2*** obtendremos el siguiente resultado.





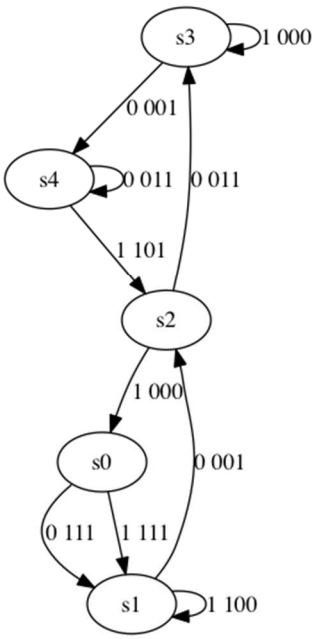
En el siguiente caso, cuando intentamos generar una máquina secuencial con un porcentaje de ciclos y saltos superior al 100% en su suma nos salta el siguiente mensaje.



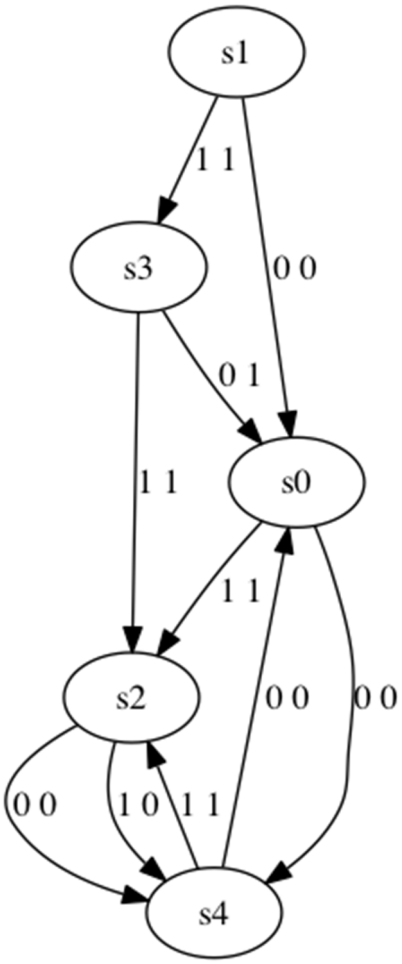


Una vez que los datos se han introducido correctamente podemos generar varios tipos de máquinas secuenciales, la de la izquierda es una máquina 100 secuencial sin condiciones de salto.

Mientras que la de la derecha es una máquina con saltos y ciclos.



Por último podremos generar la máquina de estados por patrones, la cual nos devolverá los patrones que es capaz de identificar.



En este caso el sistema devuelve:

1

01

Siendo s0 el estado inicial.

Nótese que el programa es inteligente y jamás devolvería nada de los estados s1 y s3. Dado que esta máquina no puede ir a ellos si el estado de reset o inicio es s0.

Bibliografía

1. **Commons, Creative.** Licencia Creative Commons. [Online] 2015. https://creativecommons.org/.

2. **MIT.** http://web.mit.edu/. [En línea] opensource.org/licenses/MIT.

3. **Wikipedia.** https://es.wikipedia.org/. [En línea] 2015. https://es.wikipedia.org/wiki/Licencia_MIT.

4. **Sevilla, Universidad.** personal.us.es. [En línea] http://personal.us.es/raouf/.

5. **JetBrains.** https://www.jetbrains.com/. [En línea] https://www.jetbrains.com/pycharm/.

6. —. https://www.jetbrains.com. [En línea] https://www.jetbrains.com/pycharm/features/editions_comparison_matrix.html.

7. **Git.** https://git-scm.com/. [En línea]

8. **GitHub.** github.com. [En línea]

9. **Python.** https://www.python.org/. [En línea]

10. **Pip.** https://pypi.python.org/pypi/pip. [En línea]

11. **8, PEP.** https://www.python.org/dev/peps/pep-0008/. [En línea]

12. **Tkinter.** https://wiki.python.org/moin/TkInter. [En línea]

13. **Dot, GraphViz.** http://www.graphviz.org/. [En línea]

14. **TDD.** http://agiledata.org/essays/tdd.html. [En línea]

15. **LGSynth.** http://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth89.txt. 1989.

Terminología

Benchmark: técnica o serie de metologías para determinar la eficiencia y rendimiento de un determinado componente electrónico. Así como proceso útil y necesario para hacer comparativas entre diversos componentes con un mismo propósito.

Brew: herramienta productiva para sistemas Mac OS que gestiona paquetes por el usuario.

CLI: proviene de las siglas (Client Line Interface), que en español viene a traducirse como interfaz de línea de comandos. Es una herramienta vía consola que nos soluciona un determinado problema específico.

EDA: proveniente de las siglas inglesas (Electronic Design Automation), es un tipo de software que sirve para diseñar componentes electrónicos.

GUI Client: herramienta muy parecida a una CLI pero con una interfaz más enriquecida.

PyPI: es el repositorio de paquetes de Python.

Widget: herramienta de uso frecuente y generalmente optimizado usado para el desarrollo y que cumple con un determinado patrón.