



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



TFG del Grado en Ingeniernformca
tlo del TFG



Presentado por Nombre del alumno
en Universidad de Burgos — 25 de junio
de 2019

Tutor: nombre tutor



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. nombre tutor, profesor del departamento de nombre departamento, a de nombre a.

Expone:

Que el alumno D. Nombre del alumno, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniernformca titulado tlo de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la direccil que suscribe, en virtud de lo cual se autoriza su presentaciefensa.

En Burgos, 25 de junio de 2019

V. B. del Tutor:

V. B. del co-tutor:

D. nombre tutor

D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentacil tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice general	III
Índice de figuras	V
Índice de tablas	VI
Introducción	1
Objetivos del proyecto	3
2.1. Objetivos del Proyecto	3
Conceptos teóricos	5
3.1. Fundamentos Básicos	5
3.2. De código a Tiempo Computacional	7
3.3. Tipos de Análisis	12
3.4. Estructura interna	14
3.5. Caso práctico de un Análisis	15
Técnicas y herramientas	17
4.1. Metodología Ágil	17
4.2. Herramienta de documentación	18
4.3. Herramienta de Gestión	19
4.4. Herramientas Proyecto	20
4.5. IDE	22
Aspectos relevantes del desarrollo del proyecto	23
5.1. Conceptos usados aprendidos en la Universidad	23
5.2. Transcurso del desarrollo	23

Trabajos relacionados	33
Conclusiones y Líneas de trabajo futuras	35
7.1. Conclusiones	35
7.2. Trabajos de Futuro	36

Índice de figuras

3.1. Flow diagram to transform the high-level code to computational time	7
3.2. Ejemplo de un código de alto nivel	8
3.3. Ejemplo del ByteCode en vista ensamblador	8
3.4. Formula aplicada para hacer la ponderación	12
3.5. Gráfica resultante del análisis individual	13
3.6. Estructura interna de la herramienta	14
4.7. Imagen de la interfaz de Spyder	22
5.8. Ventana principal V.1 Interfaz	27
5.9. Ventana principal V.2 Interfaz	28
5.10. Ventana principal V.3 desconfigurada	30

Índice de tablas

3.1.	ejemplo de la estructura básica del diccionario de operaciones.	. .	10
3.2.	Ejemplo de la estructura básica de la Matriz de traducción.	. . .	12
5.3.	Ejemplo de la estructura que tenía la matriz de operaciones.	. .	26

Introducción

La eficiencia en el mundo informático, es un termino fundamental y con mucho peso, que condiciona en la gran mayoría de ocasiones todo lo relacionado a este. Cuanta mayor eficiencia menos recursos se gastan para poder realizar un objetivo y estos recursos pueden ser utilizados para otras metas. La búsqueda de una buena eficiencia tiene una gran importancia dentro del ámbito laboral, donde cada recurso ahorrado puede ser fundamental, para el desarrollo de otras tareas. Un gran ejemplo de esto son los sistemas de tiempo real, donde el tiempo de respuesta debe ser mínimo, lo que hace del tiempo un recurso muy valioso, por lo cual la eficiencia de este recurso es esencial

En este caso nos vamos a centrar especialmente en la eficiencia aplicada a código Python, ya que es el tema principal de este trabajo y la eficiencia, es algo que se puede aplicar a casi cualquier campo conocido. Para poder medir la eficiencia de un código hecho en lenguaje Python, primero se tendrá que empezar por determinar y desarrollar que es lo que hace que un código sea mas eficiente que otro y el como conseguir esa información.

A la primera pregunta la respuesta seria, "depende". Ya que según el entorno y el objetivo para el que ha sido creado un programa, lo que determina si un programa esta hecho de forma eficiente puede variar entre diferentes métricas.

He aquí algunas de ellas:

- Medir el tiempo de ejecución: Esta es la forma de medir la eficiencia

mas utilizada, esto se debe a la facilidad de llevarla a cabo, pero como gran inconveniente se podría decir que es una manera la cual devuelve resultados difícilmente repetibles. Esto es por la gran influencia que tienen algunos factores externos en los resultados que puede devolver. El mayor de estos factores seria las condiciones de pruebas donde se ejecuta el código del programa, ya que si hacemos una ejecución en un mismo ordenador pero en momentos diferentes, es muy probable que algoritmos con una cierta complejidad tarde mas en ser ejecutados en un momento que en el otro. Esto se debe a Con este método para poder sacar resultados con una gran fiabilidad habría que tener un gran control sobre el entorno, lo cual requiere de una gran preparación previa.

- Complejidad O: Esta forma realmente nos sirve para conocer la complejidad del código, pero también es a veces es tomado como referencia para hacerse una estimación de la eficiencia. Al ser tan solo una estimación no se puede realizar un análisis en profundidad sobre este aspecto.
- Cuenta de Operaciones: Esta métrica se basa en contar el numero de operaciones que son necesarias en la ejecución del código. No es tan fácil de llevarla a cabo como la métrica de medir el tiempo, pero a cambio los resultados son repetibles.

La métrica elegida para ser implementada en la herramienta de este trabajo a sido la cuenta de operaciones

Se a decidido implementar la medición con operaciones por ser una manera de obtener soluciones invariables sea cual sea el entono en el que se haga y devuelve datos con una cierta profundidad como para poder hacer todos los análisis que se requieran.

En el caso de este trabajo se ha decidido que para conseguir la información de cuantas operaciones hay en la ejecución de un programa se utilizara un interprete, que como bien su nombre indica, sera el responsable de interpretar el código que forma el programa y tras hacerle unas modificaciones nos guardara la información que deseada.

Objetivos del proyecto

2.1. Objetivos del Proyecto

Objetivo general

Desarrollar un software capaz de dar la información necesaria para realizar un análisis de eficiencia sobre un código de extensión .py.

- Poder hacer análisis de un código python a través del cálculo de operaciones.
- Mostrar esa información de una manera agradable y entendible.

Objetivos funcionales

- El usuario podrá elegir el tipo de análisis que quiera realizar: de un fichero individual o comparando varios ficheros.
- El usuario podrá modificar la información resultante, eligiendo las operaciones que saldrán.
- El usuario puede cambiar la métrica de los análisis, decidiendo como pondera cada tipo de operación
- El usuario podrá realizar varios análisis sin necesidad de salir de la herramienta
- (Exportar la información?)

Objetivos técnicos

- Obtener el numero de operaciones a través del análisis por parte del interprete
- Diseñar una manera de guardas los niveles de eficiencia de cada tipo de operación.
- Obtener los datos finales con los que realizar la gráficas a través de la ponderación
- Utilizar Tkinter para realizar la interfaz de la herramienta
- Utilizar Matplotlib para generar las gráficas tras los respectivos análisis
- funcionamiento de ByteCode en Python

Conceptos teóricos

Algunos conceptos teóricos de L^AT_EX¹.

3.1. Fundamentos Básicos

La eficiencia

La eficiencia vista desde un ámbito general se definiría como el uso razonado de los recursos para conseguir un objetivo x . Esta definición puede ser aplicada a cualquier ámbito en el que se hable de eficiencia pero en el caso de este proyecto se aplicara esta definición en el ámbito computacional. La definición citada anteriormente menciona la palabra recursos, en el ámbito que estamos tratando, cuando se habla de recursos la palabra que más suele asociarse es tiempo. Esto no quiere decir que sea la única medida de eficiencia, ni mucho menos, también se podría poner como recurso el espacio en memoria, pero el tiempo suele ser el recurso que más condiciona y la gente más en cuenta tiene en las características de los programas.

Aquí llegamos a la razón de existir de este proyecto, cómo medimos lo que tarda un programa en ejecutarse de manera fiable? La manera más extendida para hacer esto, es ejecutar el programa que se desee y esperar a que termine, contado el tiempo pasado desde el comienzo hasta el final de este. Esta manera a pesar de ser sencilla, a la hora de la verdad es muy poco fiable. Para que los resultados obtenidos por este método sean considerados verídicos se tendría que asegurar que el entorno de pruebas tenga unas condiciones idóneas, como por ejemplo que el procesador que del ordenador

¹Créditos a los proyectos de Álvaro López Cantero: Configurador de Presupuestos y Roberto Izquierdo Amo: PLQuiz

donde se ejecuta el programa no este gastando recursos en otros procesos que estén activos en segundo plano. El problema de esto es que conseguir un entorno así muchas veces no es tarea fácil. Y como respuesta a este problema surge la idea principal de este proyecto, medir la eficiencia de un programa de una manera fiable, que no se vea afectado por factores externos a este. Para lograr un método métrico para la eficiencia de un programa sin que su valor se vea afectado por el cuándo y dónde se haga el análisis, se decidió hacerlo a través de las operaciones que se ejecutan en el código del programa. Tras esto una vez se obtengan estas operaciones, se hará una ponderación a cada tipo de operación según el nivel de complejidad que se les declare.

El interprete

Un interprete es un programa que analiza y ejecuta el código de otro programa instrucción a instrucción. Esta es la manera con la que se pueden detectar los diferentes elementos que componen un programa. Al realizar este trabajo en Python hay que comentar antes que el interprete se trata de una maquina de pila, lo que no deja de ser una maquina virtual que emula el funcionamiento de un ordenador. El funcionamiento principal de este se basa en almacenar y gestionar información que va recogiendo en las diversas pilas que tiene, para de esta manera poder hacer sus operaciones.

El interprete elegido para esta practica se trata de ByteRun, un interprete hecho en código python y para código python. Este fue elegido por ser un interprete que fue creado con el objetivo principal el aprendizaje, lo cual conlleva que su complejidad no sea muy alta y por lo tanto es relativamente fácil implementar nuevas funcionalidades en este. Por otro lado en contra parte de esa baja complejidad hay que mencionar que ByteRun no es un intérprete muy rápido, como por ejemplo podría ser Cpython, pero como en este trabajo el principal requisito que se busca en un interprete es el de poder implementar nuevas funcionalidades en este para conseguir una forma de poder detectar las operaciones ejecutadas y guardarlas. El ByteRun era una opción muy acertada que se adaptaba perfectamente a los requisitos del desarrollo.

Bytecode

. El ByteRun al ser un interprete de Python, no analiza directamente el código de alto nivel, lo que este analiza es un código intermedio llamado Bytecode.

El ByteCode se trata de un código intermedio sacado a través de la traducción de un código de alto nivel al ser tratado por una serie de procesos

(lexer, compiler...). Para sacar este código intermedio se ha utilizado un modulo de Python que actúa de desensamblador y descompone el código de alto nivel.

3.2. De código a Tiempo Computacional

Una vez explicados los conceptos anteriores se para a contar los pasos llevados a cabo para conseguir traducir el código de alto nivel a una medida con la que se pueda medir la eficiencia, el cual en el caso de este trabajo se ha optado por los ciclos de reloj.

Los pasos para hacer la transformación son los siguientes:

1. Transformar el código de alto nivel a ByteCode a través del compilador (dis).
2. El interprete traduce y ejecuta el ByteCode.
3. Mientras el interprete ejecuta el ByteCode, se va obteniendo el diccionario de operaciones.
4. Hacer la ponderación de los valores recogidos en el diccionario de operaciones con los procesadores teóricos para obtener el tiempo computacional.

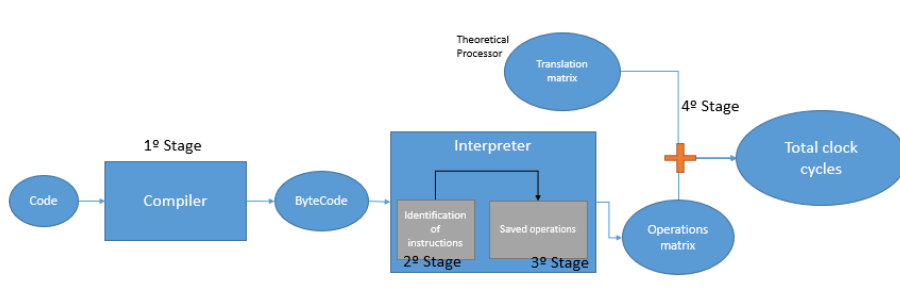


Figura 3.1: Flow diagram to transform the high-level code to computational time

1º Paso: Obtención del ByteCode

Como ya se ha comentado antes el ByteCode se trata de un código intermedio sacado a partir de la intervención de un compilador, para conseguir esto se cuenta con un modulo de Python llamado Dis que hace esta labor de compilador.

De primeras el aspecto de este código se trata de una serie de bytes que a simple vista una persona no podría entender.

```
x=10
y=8
z=x*y
```

Figura 3.2: Ejemplo de un código de alto nivel

Pero el modulo Dis, cuenta con una función que permite ver el ByteCode en una especie de lenguaje ensamblador, lo cual resulta muy útil para entender la estructura del código.

1	0	LOAD_CONST	0 (10)
	3	STORE_NAME	0 (x)
2	6	LOAD_CONST	1 (8)
	9	STORE_NAME	1 (y)
3	12	LOAD_NAME	0 (x)
	15	LOAD_NAME	1 (y)
	18	BINARY_MULTIPLY	
	19	STORE_NAME	2 (z)
	22	LOAD_CONST	2 (None)
	25	RETURN_VALUE	

Figura 3.3: Ejemplo del ByteCode en vista ensamblador

La forma de entender este tipo de código es la siguiente:

Como ejemplo usaremos la primera fila donde se encuentra la instrucción "LOAD CONST", la primera columna muestra el número 1 que representa la

línea donde está la instrucción en el código de Python, la segunda columna es un índice que indica que la instrucción "LOAD CONTS", en En este caso, está en la posición 0, la tercera columna es el nombre de la instrucción en sí misma, mostrándola con un nombre comprensible para una persona, la cuarta columna indica la posición de ese argumento en la pila y la última columna muestra cuál es el argumento para la instrucción.

2º Paso: Obtención del ByteCode

. Una vez obtenido el Bytecode entra en , el interprete, en nuestro caso el ByteRun, es capaz de ir analizando y ejecutando el bytecode obtenido. .

3º Paso: El Diccinario de procesos

. El ByteRun no es capaz ir almacenando las operaciones que va traduciendo. En este momento es donde se debe implementar una nueva funcionalidad en el interprete para que sea capaz de hacer esto, para así mas tarde poder hacer un calculo de la eficiencia

Cuando el ByteRun lee una instrucción y lo primero que tiene que hacer es identificar que tipo de instrucción esta leyendo, hay muchos tipos diferentes desde un "LOAD", a un "LOOP", o un ".ADD", pero a nosotros no nos interesan todas las instrucciones, solo queremos almacenar aquellas que son consideradas por nosotros como ".operaciones", o lo que es lo mismo, instrucciones que necesiten de algun tipo de calculo por parte del procesador, y que por lo tanto tenga un tiempo de ejecución.

Por suerte para nosotros el interprete ya hace distinciones entre los diferentes tipos de instrucciones y guarda en una lista todos aquellos valores en los cuales requiere hacer una operación.

Aprovechando esto, cada vez que alguno de esas instrucciones es llamada guardamos el nombre que que tiene esta al ser visualizado el bytecode mediante el modulo dis en un diccionario, el cual decidimos llamar diccionario de operaciones.

Este diccionario tiene la siguiente estructura:

Operación	Contador
ADD int int	X
ADD int flo	X
MULTIPLY int int	X
MULTIPLY str int	X
DIVIDE int int	X

Tabla 3.1: ejemplo de la estructura básica del diccionario de operaciones.

Se trata de un diccionario compuesto por una lista de clave primaria, y los componentes de esta lista son los siguiente:

- Nombre de la operación detectada por la interfaz
- Primeras 3 Iniciales del tipo del primer operador
- Primeras 3 Iniciales del tipo del segundo operador

Esta estructura para las keys del diccionario se basa en el hecho de que un operación necesita de dos valores con los que operar, y dependiendo de el tipo de estos valores, dos mismas operaciones con diferentes valores, pueden llegar a tener nivel de eficiencia mucho mas bajo o alto que el otro.

Según el ByteRun va analizando todo el código cuando detecta una instrucción dentro de la lista de operaciones este llama a una función, que en caso de no tener ninguna entrada de esa operación y sus respectivos operadores la genera y pone de valor un 1. Posteriormente si se va encontrando mas operaciones iguales en vez de crear otra entrada en el diccionario de operaciones, incrementa en 1 el valor enlazado con la clave que la corresponde.

De esta manera cuando el ByteRun acaba de analizar todas las operaciones, conseguimos tener un diccionario con todas las operaciones que han surgido a lo largo de la ejecución del programa.

4º Paso: Ponderación

Pero con solo tener los tipos de operaciones no es suficiente. Para hacer un análisis de la eficiencia es necesario saber el nivel de eficiencia que dispone cada tipo de operación, ya que no todas las operaciones tiene la misma complejidad y el procesador necesita hacer mas cálculos para ejecutarlos. Esto se consigue gracias a el procesador teórico.

Procesadores Teóricos

El procesador teórico tiene la función de darle un nivel de eficiencia a cada tipo de operación. Se le puso este nombre por ser la parte que simula la complejidad que le supondría a un determinado procesador, de esta manera según los valores que se guarden en el procesador se pueden simular diferentes tipos de entornos.

Estos procesadores se tratan de unos ficheros de tipo .csv que guardan una matriz, llamada matriz de traducción, la cual esta hecha de tal tiene una estructura pensada para poder sacar mas los niveles de eficiencia que guarda en cada celda según sea necesario.

La dos primeras celdas de la matriz hacen de indice para las tuplas, la primera columna guarda el tipo de operación mientras que la segunda guarda el tipo del primer operador. Siguiendo esta estructura si tuviéramos por ejemplo una suma de un entero con un decimal, para encontrar el nivel de eficiencia que habría que aplicarle a esta operación en concreto la herramienta busca primero el tipo del operador, en este caso seria `.^DDz` seguido a esto busca en la segunda columna deja el valor `int`, una vez encontrados estos dos valores ya tendría localizada la tupla donde se encuentra el valor deseado, y aquí es donde entra en juego el indice de las columnas que se haya en la primera fila de la matriz, esta guarda los diferentes tipos que puede tomar el segundo operador, en este caso `"flo"`, y con esto puede localizar que celda de la tupla es la que se necesita.

Calculo de la Ponderacion

Una vez tenemos tanto el diccionario de operaciones como un Procesador teórico ya solo falta realizar la ponderación para obtener los datos necesarios, en este caso los ciclos de reloj, para hacer los análisis oportunos.

La formula aplicada para hacer la ponderación es la siguiente:

Operación	Operador	INT	STR	FLOAT	BOOL
ADD	int	1	4	2	3
ADD	int	2	2	2	2
MULTIPLY	int	1	2	3	2
MULTIPLY	int	1	4	2	2
DIVIDE	int	2	5	3	3

Tabla 3.2: Ejemplo de la estructura básica de la Matriz de traducción.

$$\mathbf{T}_{cloc} = \sum_{(i,j)}^{(N,M)} M_O(i,j) M_T(i,j)$$

Figura 3.4: Formula aplicada para hacer la ponderación

Donde N y M son el número de filas y columnas de M O. Debe prestarse atención al tamaño de M T debe ser más grande que el tamaño de M O. Aunque esta herramienta se ha desarrollado completamente en Python, y actualmente solo puede analizar archivos .py exclusivamente, una de las ideas principales es que luego se pueda usar en diferentes entornos, para diferentes tipos de lenguajes y pueda realizar una variedad más amplia de análisis, para así adaptarse mejor a las diferentes demandas que puede requerir el análisis en profundidad de la eficiencia de un código, en un entorno más exigente. Se decidió comenzar con este tipo de lenguaje, ya que Python hoy en día se usa ampliamente en diferentes campos como por ejemplo la robótica, gracias a la gran variedad de bibliotecas que tiene. Y sería muy interesante ver los resultados que se podrían obtener en todos estos campos.

3.3. Tipos de Análisis

Con los datos obtenidos ya se pueden hacer diferentes tipos de análisis, en el caso de esta herramienta se han implementado dos tipos:

- Análisis individual
- Análisis comparativo

Analisis Individual

En este análisis solo se permite la entrada de un fichero, de formato .py, el cual tras pasar por los pasos citados en los puntos anteriores, muestra una gráfica circular con las diferentes operaciones que el interprete(ByteRun) a detectado al ejecutarlo. Y además de esto deja recalculare el resultado mostrado en el gráfico según las operaciones que queramos mostrar o no y el procesador con el que se quiera hacer la ponderación.

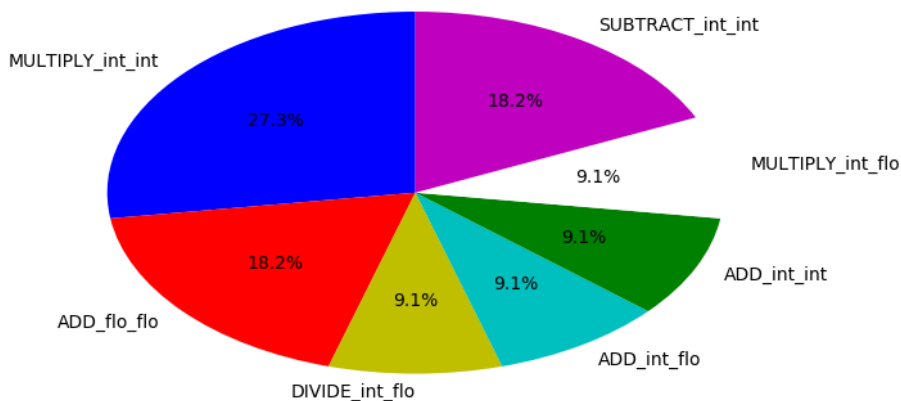


Figura 3.5: Gráfica resultante del análisis individual

Este análisis está pensado para optimizar el código del programa, ya que de esta manera puedes detectar que operaciones son las que más recursos consumen.

Análisis Múltiple

En este análisis se deben elegir más de un fichero, los cuales ejecuta y calcula los ciclos de reloj que tarda en ejecutarse cada uno, pero al mostrar los resultados varía según el número de ficheros que se hayan escogido. En caso de ser menos de 10 saldrá una gráfica de barras en la cual cada barra representa el total de ciclos de reloj de cada fichero. Y en caso de que se metan más de 10 se cambiaría por una gráfica de puntos para facilitar la visibilidad de los resultados. Al igual que en el análisis individual en todo momento se pueden cambiar los resultados de la gráfica cambiando las operaciones que queramos que se tengan en cuenta y el procesador con el

que se quiera ponderar las operaciones.

Imagen resultados:

Este análisis esta pensado para compara ficheros que tengan como finalidad hacer lo mismo, y de esta manera saber cual de ellos tiene en que clase de entornos mejor eficiencia.

3.4. Estructura interna

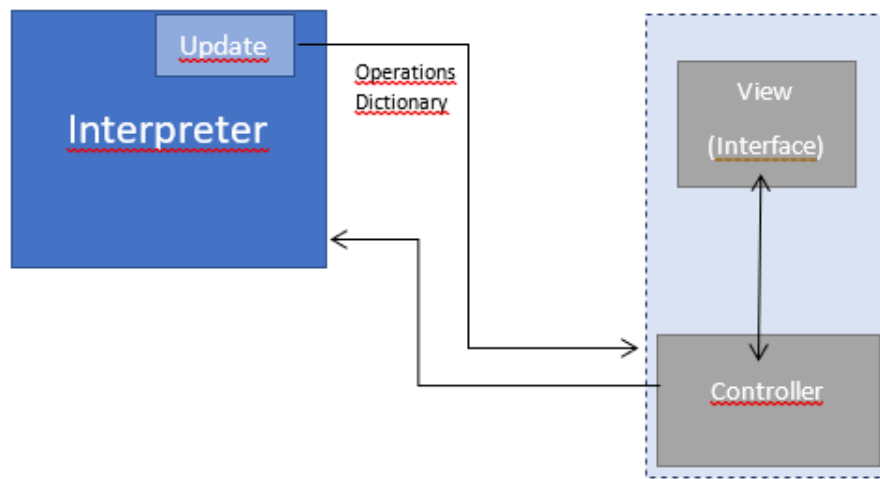


Figura 3.6: Estructura interna de la herramienta

La estructura interna del programa se muestra en la figura 4. Como se puede observar en la figura 4, es un modelo vista controlador . En este caso, el controlador y la vista se implementan de tal manera que comparten algunos componentes, por lo tanto, comparten un cuadro en la imagen, pero aún así, el concepto del modelo es el típico en el que el controlador actúa como un intermediario. Esta unión surgió del desarrollo, se fueron haciendo partes del controlador junto con la interfaz.

En la herramienta, el controlador actúa como un intermediario entre la interfaz gráfica y el interprete. La interfaz permite interactuar y mostrar los resultados del análisis de eficiencia, mientras que el intérprete, el ByteRun en este caso, se ocupa de analizar y ejecutar el código.

3.5. Caso práctico de un Análisis

Técnicas y herramientas

4.1. Metodología Ágil

Para mantener un flujo de trabajo en el desarrollo del proyecto se decidió utilizar la metodología Kanban para así mantener un cierto progreso en todo momento.

Kanban

La metodología Kanban se basa en hacer visible el flujo de trabajo a través de una tabla. La tabla Kanban se puede dividir en diferentes filas y columnas, las filas sirven para identificar diferentes tipos de actividades y las columnas para identificar cada paso por un proceso.

En el caso de la tablas que se ha realizado para este proyecto solo se ha utilizado una fila, ya que al solo tener 2 actores en el proyecto (Mi tutor y Yo), identificar el tipo de actividades no se creyó necesario. Por otra parte se implementaron 4 columnas:

- Por hacer.
- En proceso.
- Revisión.
- Hecho.

(Imagen de tabla)

Con esta tabla el sistema de trabajo ha sido el siguiente:

Primero se hacia una reunión con una frecuencia de una o dos semanas, aquí se planteaba que tareas había que hacer. Estas tareas eran las que se colocaban en la columna "Por hacer", una vez hecho esto las tareas que se empezaban a desarrollar pasaban a ^{En} "Proceso", con un limite de 2 tareas simultaneas en esa columna, no se pueden estar desarrollando mas de dos tareas a la vez. Esto se decidió así para que no se concentrase tanto el flujo en ese proceso. Una vez una tarea se acababa su desarrollo pasaba a Revisión, lo que significaba que hasta que en la siguiente reunión no se le diese el visto bueno por parte del tutor, no pasaba a Terminado.

4.2. Herramienta de documentación

Latex

Es un sistema de composición de textos especializado en los textos técnicos y científicos. Se decidió hacer este trabajo con este sistema para componer la documentación, porque facilita de gran manera la buena estructuración del documento, gracias a su , además de ofrecer una gran cantidad de aspectos tipográficos, que consiguen dar una gran calidad y profesionalidad a los documentos resultantes y al componer Latex texto mediante marcas en un archivo fuente, esto permite previsualizar el documento desde cualquier entorno sin perder el formato, lo resulta extremadamente útil para el desarrollo de documentos como el que estas leyendo.

En mi caso he utilizado como editor de texto TexMaker, ya que me gustaba las disposición en la que dispone su interfaz y me facilitaba mucho su uso.(Enlace pagina de TexMaker)

Version: 5.0.3

Licencia Open Source

PaginaWeb: <https://www.xmlmath.net/texmaker/download.html>

La distribución de Latex utilizada a sido miktex, ya que esta contiene un gran número de paquetes tipográficos y es fácil de instalar:

Version: 2.9.6

Licencia Open Source

PaginaWeb: <https://miktex.org/download>

4.3. Herramienta de Gestión

GitHub

Se trata de una plataforma online donde la gente puede almacenar y gestionar los proyectos que estén desarrollando mediante gestión de versiones.

Ha sido la herramienta de gestión principal elegida debido a que es la que nos han enseñado su uso en la Universidad y por lo tanto con la que más practica se tenía de antemano, además de ser gratuita para proyectos de código abierto.

Pagina web: <https://github.com/>

La hemos utilizado para albergar el código del proyecto, y desde ahí poder gestionar el avance del desarrollo.

Link Repositorio: https://github.com/Guillecal/TFG-Herramienta_para_medir_la_eficiencia_de_codigo_python

Se han ido añadiendo los commits pertinentes según se han ido completando las tareas planificadas.

GitHub Desktop

Esta es la herramienta utilizada para gestionar de mejor manera el repositorio donde se encuentra el trabajo, ya que facilita mucho el poder realizar los commits, con sus respectivos comentarios e incluso dejar gestionar su contenido.

Pagina web: <https://desktop.github.com/>

Una vez descargado e instalado, la primera vez que lo ejecutamos, podemos clonar desde aquí el proyecto desde el repositorio.

4.4. Herramientas Proyecto

Python

Se a utilizado este lenguaje de programación para realizar el trabajo, esto se debe a las gran cantidad de librerías que dispone este lenguaje para poder operar con una gran variedad de componentes, lo que ya de por si nos ofrece mucha flexibilidad. También hay que destacar que se elgio este lenguaje por ser un tipo de lenguaje interpretado, lo cual hacia que fuese mas facil tratar el tema del interprete. La versión con la que se ha contado a lo largo del proyecto ha sido la 2.7.16. Se intento utilizar un versión 3.7 y 3.6, pero tras tener varios problemas de consistencia entre este y el interprete que elegimos para la realización del proyecto, al final esto nos forzó a tener que probar una versión 2.

Version:2.7.16

Licencia: Open Source

PaginaWeb: <https://www.python.org/downloads/>

Tkinter

Es una librería de Python para el desarrollo de interfaces gráficas. En principio se barajó la posibilidad de utilizar algún otro tipo de librería para realizar la interfaz como por ejemplo C, pero al tener dificultades de implementar otras librerías la versión de Python 2.7., y no tener Tkinte una gran dificultad de aprendizaje para poder realizar una interfaz básica.

Version: 8.5

Licencia: Open Source

PaginaWeb:

Se utilizo un modulo del Tkinter ttk

MatPlotLib

Es una librería diseñada para representar datos en gráficos 2D. Es una librería clave en el proyecto para la representación de los gráficos resultantes de los análisis.

Version: 2.2.4

Licencia: Open Source

PaginaWeb: <https://matplotlib.org/downloads.html>

Microsoft Excel

Es una aplicación de hoja de cálculos, utilizada mayoritariamente para tareas financieras y de logística. En el caso de este proyecto se utilizó únicamente para dar valores a las matrices de traducción, ya que la estructura de celdas que tiene esta aplicación facilitaba mucho hacer esta tarea.

Version: 1905, Office 2016

Licencia: Comercial, Propietario

PaginaWeb: <https://products.office.com/es-es/try>

Csv

Módulo de Python que permite abrir, leer y escribir en ficheros .csv. En el caso de este trabajo ha sido utilizado para poder leer los procesadores teóricos y poder sacar los valores deseados para realizar la ponderación.

os

Se trata de un módulo de Python, que permite acceder a funciones que permiten leer y escribir archivos y acceder al sistema de archivos. En el caso de este proyecto se ha utilizado para acceder al sistema de archivos y poder recoger las rutas de algunas carpetas de forma automática.

Dis

Un módulo de Python que desensambla el código origen y lo transforma a ByteCode descomponiendo cada parte del código original. Este módulo es vital para el funcionamiento del ByteRun ya que es el encargado de transformar el código de alto nivel a ByteRun

4.5. IDE

Como entorno de desarrollo se podría utilizar el IDE de Python, pero para el desarrollo de la herramienta se ha utilizado Spyder.

Spyder

Spyder es un entorno de desarrollo diseñado para desarrollar código Python, el cual cuenta con un terminal donde se pueden probar los códigos y ya incluye algunas de las librerías mas utilizadas de Python

Link descarga: <https://www.spyder-ide.org/>

Una vez descargado e instalado este IDE ya viene listo para poder empezar a trabajar, pero en caso de que se requiera, se pueden cambiar algunas configuraciones según el gusto de cada uno desde la pestaña Herramientas, en el apartado preferencias.

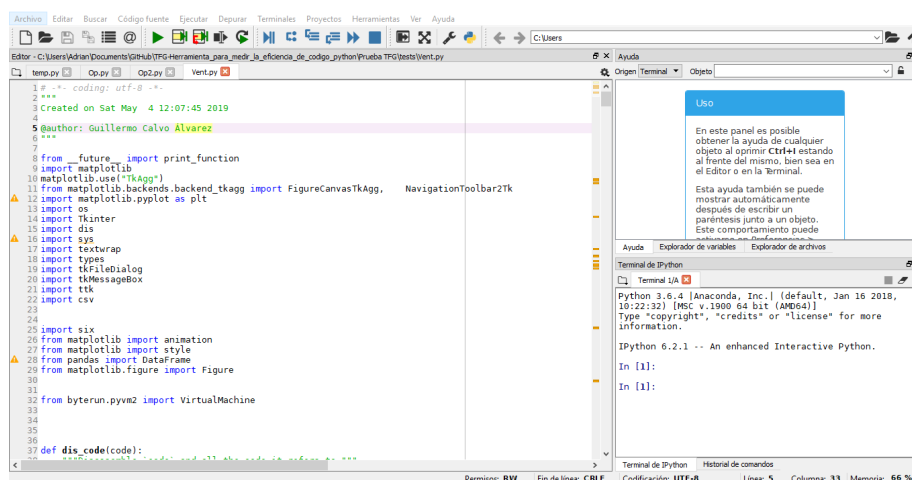


Figura 4.7: Imagen de la interfaz de Spyder

Esta es una recomendación, si se está acostumbrado utilizar otro tipo de IDE que sirva para el lenguaje Python no habría ningún problema.

Aspectos relevantes del desarrollo del proyecto

5.1. Conceptos usados aprendidos en la Universidad

5.2. Transcurso del desarrollo

El planteamiento inicial del proyecto era crear una herramienta para medir la eficiencia del código Python, en base a los ciclos de reloj y el espacio en memoria que este ocupa. Al ser este un planteamiento previo al inicio del desarrollo, este ha ido dando algunos cambios según nuevos requisitos han ido surgiendo o restricciones con las que no habíamos contado han ido apareciendo. En este apartado se intentara contar como han surgido esos cambios inesperados y como han afectado este al desarrollo del proyecto.

El interprete

En primer lugar, la base por donde se partió el proyecto fue el intérprete. Se buscó un intérprete capaz de satisfacer las necesidades que se plantearon. En esta etapa surgieron varias propuestas, pero tras algunas puestas en común, se llegó a la conclusión de que el intérprete ByteRun era la mejor idea para hacer un primer intento.

Requisitos que cumple

ByteRun al final fue el intérprete elegido porque nos proporcionaba una serie de requisitos que encajaban a la perfección con lo que teníamos planeado previamente. He aquí alguno de esos requisitos:

- Estaba desarrollado en Python: Al ser Python un lenguaje conocido y que ha sido impartido en la Universidad, era preferible que estuviese hecho en un lenguaje que ya tuviésemos conocimientos previos. Además que un intérprete de Python hecho en Python nos parecía que tenía un cierto atractivo.
- Los autores de este interprete contaban con una buena documentación sobre su funcionamiento, lo cual nos venía como anillo al dedo, para así entender mejor su funcionamiento y poder hacer mejor el upgrade a este.
- Es un intérprete hecho con la finalidad del aprendizaje, esto supone que no sea el más rápido, pero también conlleva el hecho de que no sea muy complejo. Y como el objetivo del proyecto no es tener la herramienta más rápida, si no que funcione y seamos capaces de implementar los cambios necesarios, esto para nosotros era visto como una gran ventaja.

Una vez se tenía claro que el ByteRun era el intérprete que haría de base para desarrollar la herramienta, lo primero era entender su funcionamiento y hacer pruebas con diferentes ficheros para comprobar que de verdad funcionaba.

Pruebas

En este punto nos encontramos con los primeros imprevistos, los autores del intérprete ya contaban con unos tests hechos, pero para poder ejecutar estos test utilizaban una herramienta para Python llamada tox.

Tox en algunos tipos de versiones de Python no funciona adecuadamente, tras informarnos sobre esta, y hacer pruebas en diversas versiones de python, al final decidimos implementarlo en Python 2.7.16 la última de las versiones

2 de Python actualmente.

Una vez solucionado esto se probaron diferentes tests que estaban ya creados, los cuales funcionaban correctamente. Pero para poder medir los límites del interprete se generaron otra serie de tests con los cuales se pudo demostrar alguna de las limitaciones del interprete.

Upgrade

Seguido de las pruebas se pasó a plantear la modificación del intérprete. Al principio como la intención del proyecto era poder medir la eficiencia a través de las operaciones y de las direcciones de memoria, nos pusimos a investigar en que parte del intérprete conseguiríamos la información necesaria para esto, y la conclusión a la que llegamos no fue la esperada. Esto se debió a que al ser el ByteRun un intérprete de Pila, este no guarda la información que va procesando directamente en direcciones de memoria, si no que las guarda en su sistema de pilas interno. Por culpa de esto nos vimos obligados a cambiar el planteamiento principal y centrarnos en la eficiencia a través de las operaciones y diferentes formas sacar resultados a partir de esto.

Una vez centrados en sacar las operaciones del interprete, nos topamos con algo que esperábamos. El interprete para saber como actuar según el tipo de instrucción que fuera a ejecutar distinguía entre diferentes tipos de instrucción, y para las que nosotros llamaríamos operaciones, tenía un par de listas con los nombres de estas según el ByteCode. Con esta diferenciación, nosotros tendríamos que guardar una instrucción cuando esta se encuentra en alguna de estas listas.

El proceso de guardado de matrices paso por varias etapas para al final conseguir, al a nuestro criterio, el método mas optimo de entre todos ellos.

En la primera idea implementada solo se crearon unas variables por cada tipo de operación disponible, esto a parte de ser poco eficiente, limitaba mucho el tipo de operaciones a analizar, ya que no se tenían en cuenta los tipos de los operadores.

La segunda idea implementada fue la de la matriz de operaciones: se decidió guardar las operaciones en una matriz con la misma estructura que

posteriormente se aplicaría a la matriz de traducción.

Operación	Operador	INT	STR	FLOAT	BOOL
ADD	int	1300	0	0	0
ADD	int	0	2000	0	0
MULTIPLY	int	0	0	300	0
MULTIPLY	int	50	0	0	0
DIVIDE	int	250	0	355	0

Tabla 5.3: Ejemplo de la estructura que tenía la matriz de operaciones.

Esta idea ya daba una mayor variedad de operaciones, ya que si tenía en cuenta los tipos de los operadores, pero por otra parte era incluso menos eficiente que cuando solo había variables. La matriz hacia tener un gran numero de posiciones en la matriz sin ningún tipo de uso, ya que la matriz tenía unas 350 posiciones de las cuales por fichero se solían llenar entre 6 y 12. Esto equivalía un porcentaje muy grande de desuso, por lo que se decidió cambiar este método por otro.

El método que ya se decidió como definitivo, como ya se explica en el apartado de Conceptos Teóricos, se trata de un diccionario que guarda un contador por cada operación diferente, teniendo en cuenta los operadores. Y de esta manera nos ahorramos las celdas vacías que nos surgían en la matriz. (Foto diccionario otra vez?)

Análisis

Tras implementar esto empezamos a plantearnos los distintos tipos de análisis que se harían con los datos obtenidos y la conclusión a la que llegamos fue el de hacer un análisis individual en el que solo mostrásemos los porcentajes de eficiencia de un fichero dado y otro en el pudiésemos comparar la eficacia entre varios varios ficheros.

Primero empezamos por el análisis individual, y en este caso como el controlador y la vista están unidas, se podría decir que era un análisis en paralelo, según se hacia esa parte del controlador, se implementaban los

parámetros de la interfaz.

La interfaz también paso por una serie de versiones: Al principio la interfaz tan solo se componía de una ventana donde se añadían elementos y a veces se superponían, literalmente era un caos. Además de esto para indicar que fichero se quería analizar había que pasar la dirección del directorio.

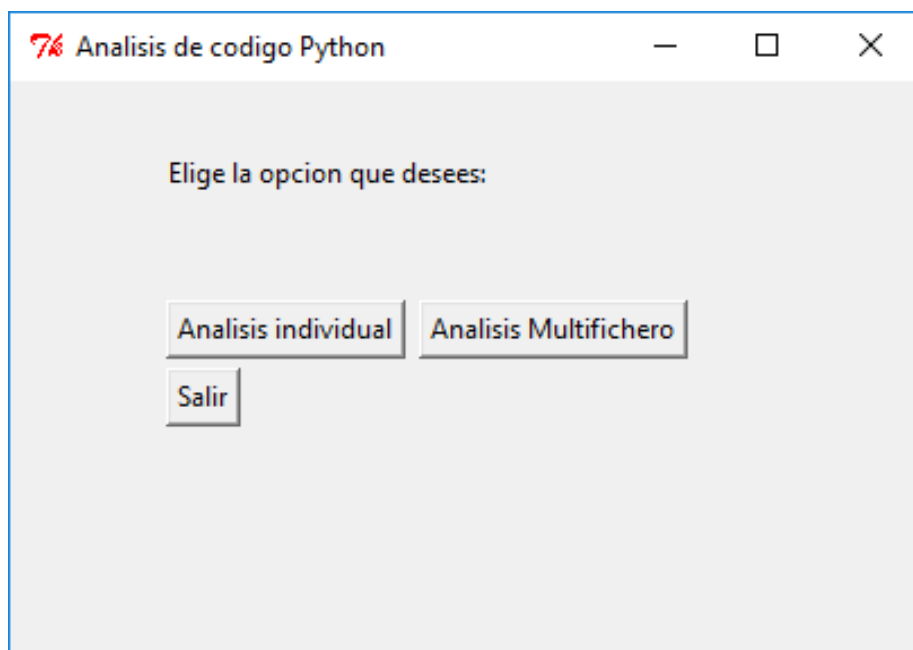


Figura 5.8: Ventana principal V.1 Interfaz

Para evitar que los elementos colisionasen entre ellos, se paso a otro método donde cada vez que se se activasen algunos botones en particular, se añadiese una nueva ventana, esto evitaba el problema de las colisiones, pero por otra parte la final de los análisis era muy molesto, porque podías terminar con 4 ventanas diferentes abiertas, lo que al final era muy lioso.

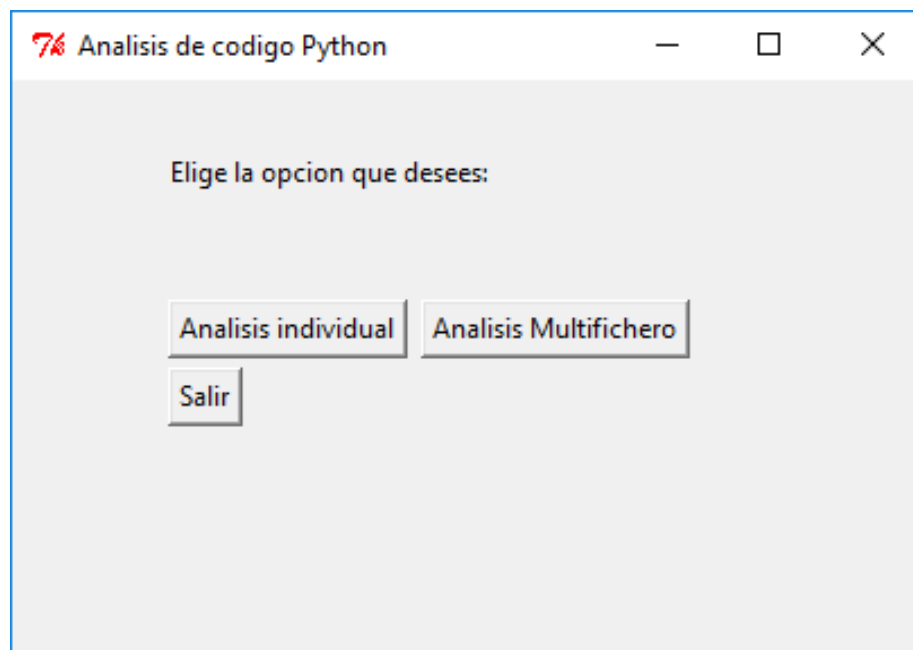


Figura 5.9: Ventana principal V.2 Interfaz

Para evitar los problemas tanto de la primera versión de la interfaz como de la segunda, a través de investigar un poco mas en el funcionamiento de la biblioteca Tkinter y sus diferentes funcionalidades, se consiguió realizar una interfaz que utilizase solo una ventana, pero el contenido de esta se fuese cambiando según interactuemos con esta. A decir verdad la interfaz por dentro crea un numero de marcos, por así llamarlos, donde ya están todos los elementos que se tiene que mostrar en pantalla, cada marco representa un tipo distinto de ventana que puede surgir según se desarrolle la interacción con la herramienta, lo único que solo se muestran por pantalla aquel marco que corresponda con la funcionalidad que se este mostrando en ese momento.

Fotos

En concreto esta herramienta cuenta con un total de 6 (A revisar) Marcos que son los siguientes:

- VentanaPrincipal
- VentanaIndividual

- VentanaAnálisis
- VentanaMultiple
- VentanaComparación
- VentanaComparación2
- VentanaOperaciones

Es una manera muy útil de hacer una interfaz mucho mas fluida y sencilla para la vista. La única pega que se le podría poner a este método, es que al iniciar la aplicación tiene que generar los marcos para ir rotando entre ellos. Esto ha condicionado más de una vez algunas implementaciones hechas en la interfaz ya que a veces quedan huecos innecesarios.

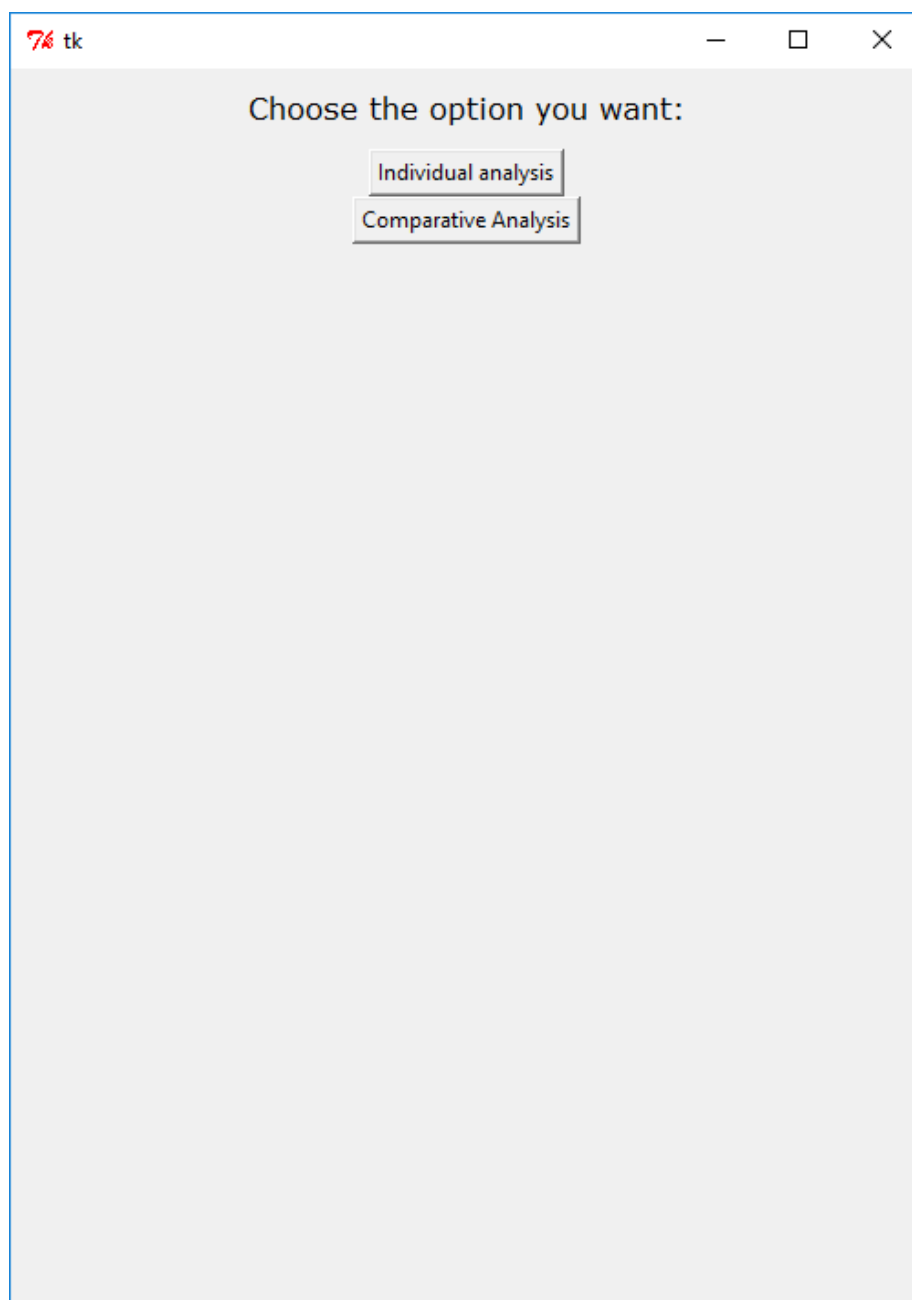


Figura 5.10: Ventana principal V.3 desconfigurada

Para acabar con el desarrollo de las funcionalidades principales se crearon los llamados procesadores teóricos a través de la matriz de traducción. Como ya se ha explicado en puntos anteriores los procesadores contienen una

matriz de traducción que contiene valores que dan un nivel de eficiencia a los distintos tipos de operaciones. Para guardar la matriz se decidió utilizar el formato csv. Hay un par de motivos detrás de esta elección:

1. La simplicidad del formato hace posible poder configurarlo con un gran numero de aplicaciones, aunque en el caso de este proyecto se opto por Microsoft Excel.
2. Hay un modulo de Python del que ya teníamos algo de experiencia, llamado también csv, que permitía leer y escribir en los tipos de archivos .csv.

A finales del desarrollo del proyecto se trato de implementar una funcionalidad mas, pero al final se decidió desechar esta, debido a que implicaba muchos mas recursos de los pensados en un planteamiento previo y para dejarlo inconcluso preferimos abortar.

Artículo

Una vez terminado el desarrollo de las funcionalidades principales se le dieron unas pinceladas a algunos aspectos de este como proporciones de algunos elementos en la interfaz, cambios en las nomenclaturas de variables, etc. Y mientras esto ocurría empezamos a desarrollar un articulo sobre el propio proyecto, para exponerlo en el congreso europeo, EuroSim. Pensamos que seria una oportunidad muy buena de ver si el trabajo echo hasta el momento era visto atractivo y útil por otras personas (Del sector.. explicar mas a fondo)

Retoques

Una vez Terminado el artículo se paso a una fase final de depuración donde se hicieron algunos cambios de diseño o alguna pequeña funcionalidad de ultima hora.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Para terminar se contarán las conclusiones obtenidas tal el desarrollo de todo este proyecto y las posibilidades que deja abierta para desarrollar más cosas en el futuro:

7.1. Conclusiones

- Se consiguieron realizar los objetivos principales propuestos para este proyecto. Entre estos estaría la implementación de una forma de medir la eficiencia de una manera flexible, con el propósito de que sea útil en más de .
- La labor de comprensión realizada para entender un código hecho por otras personas, en este caso concreto para entender el funcionamiento del ByteRun, fue más duradero de lo esperado ya que a pesar de no ser un intérprete muy complejo tenía una serie de conceptos sobre las pilas y el Bytecode, al final fue una buena manera de aprender algo más sobre los diferentes tipos de intérpretes que había y en concreto saber más sobre los intérpretes de pila.
- Otro gran reto de esta aplicación fue la interfaz gráfica, ya que casi no había tenido experiencia creando una. Tanto elegir las herramientas necesarias para hacerla como el posterior aprendizaje sobre el funcionamiento de Tkinter, es otra de las cosas que me llevo aprendidas tras este proyecto. Aunque no todas las ideas que se plantearon para este proyecto salieron finalmente a la luz, ha sido muy satisfactorio y

enriquecedor, ver como ideas que vas planteando se van plasmando poco a poco en algo que vas invirtiendo horas para finalmente formar aquello que estabas intentando lograr. Personalmente también me ha servido para aprender sobre la gestión que hay que llevar en un proyecto de estas características y como reaccionar ante los diferentes obstáculos que uno se va encontrando en el desarrollo de este.

7.2. Trabajos de Futuro

Este proyecto tiene un fuerte potencial para seguir siendo desarrollado, ya que a partir de este punto puede evolucionar hacia diferentes frentes:

En la herramienta se han implementado aquellos análisis que se han creído mas interesantes en una primera instancia que puede ser usado en una ámbito mas general. Pero se pueden implementar mas análisis dependiendo del objetivo que se tenga detrás, por poner un ejemplo, un análisis que estuvo a punto de ser implementado también en este proyecto fue, el análisis de operaciones individuales. Este análisis consistiría en mostrar como aumenta la cantidad de un tipo de operador en concreto cada vez que el interprete encuentra un operador. Esta seria una buena forma de ver en que parte de la ejecución un código se aglomeran mas algunos tipos de operaciones. Y este es solo un ejemplo de los muchos tipos de análisis mas que se pueden implementar.

Y en lo que mas potencial creemos que puede derivar en un futuro esta herramienta seria el poder medir la eficiencia de códigos de casi cualquier tipo, esto seria relativamente sencillo tras haber desarrollado esta herramienta, ya que por la forma interna en la que esta construida . La interfaz esta separada del resto del programa y gracias a esto se podrían añadir interpretes que pudieran analizar tipos de lenguaje de programación, y al resto de la herramienta habría que hacerle unos cambios mínimos para que permitiese la entrada de ficheros con otro tipo de formato que no sea solamente .py. Esto haría que la herramienta fuese útil para casi cualquier tipo de desarrollo.

Como últimos planteamiento también se podría mirar la migración de version, de la version 2.7 de Python a alguna de las versiones 3. Algunos elementos de la interfaz podrían se mejorables como el evitar la superposición de los nombres de los tipos de operaciones detectados en el análisis individual cuando se trata de un fichero con muchos tipos.