**Ejercicio de ejemplo:**

**Directorio inicial de ejercicios**

dirEjercicio

---- main.py (Código base, CB)

---- s1 (Directorio de posibles variantes para el subcódigo s1 – Variantes de subcódigo, VS)

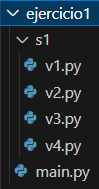
-------- v1.py (Variante 1 para el subcódigo s1)

-------- v2.py

-------- v3.py

-------- v4.py

---- descripcion.txt (Archivo de descripción del ejercicio con texto variable – Sugerido, pero no necesario)



**Detalles de los archivos de ejercicio**

main.py

A screen shot of a computer

Description automatically generated

v1.py



v2.py

A black background with white text

Description automatically generated

v3.py



v4.py

A screenshot of a computer

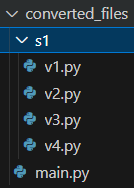
Description automatically generated

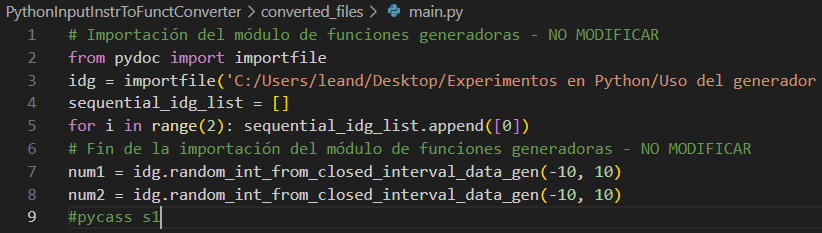
Módulo 1

A screenshot of a computer program

Description automatically generated

Archivos generados por el módulo 1





# Importación del módulo de funciones generadoras - NO MODIFICAR

from pydoc import importfile

idg = importfile('C:/Users/leand/Desktop/Experimentos en Python/Uso del generador automático de ejercicios/PythonInputInstrToFunctConverter/input\_data\_generators.py')

sequential\_idg\_list = []

for i in range(2): sequential\_idg\_list.append([0])

# Fin de la importación del módulo de funciones generadoras - NO MODIFICAR

num1 = idg.random\_int\_from\_closed\_interval\_data\_gen(-10, 10)

num2 = idg.random\_int\_from\_closed\_interval\_data\_gen(-10, 10)

#pycass s1

No hubo cambios en los demás archivos porque no había operaciones de lectura en esos códigos fuente.

Entonces, lo que se obtiene en este primer paso es la transformación de códigos fuente, reemplazando todas las instrucciones de lectura de datos por funciones generadoras que retornan un valor al ser llamadas. Existen diferentes funciones generadoras para propósitos diferentes.

Además, se añade código fuente como encabezado para los códigos que tenían instrucciones de lectura para que las funciones generadoras funcionen. El código añadido puede incluir listas (i.e., arreglos) que almacenen los estados de cada función generadora, para que estos puedan generar un valor diferente por cada llamado a la función, en el caso de que dicha función genere un patrón de valores.

Módulo 2

A screenshot of a computer

Description automatically generated

Archivos generados por el módulo 2

A screenshot of a computer

Description automatically generated

res1.py

A screen shot of a computer program

Description automatically generated

res2.py

A screen shot of a computer program

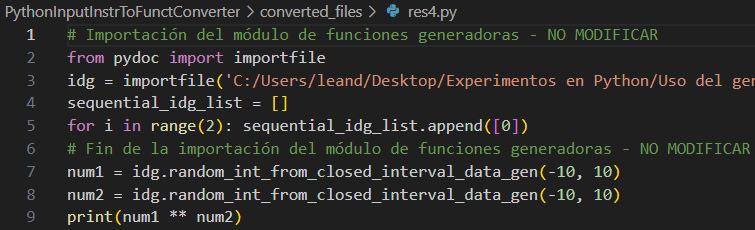
Description automatically generated

res3.py

A screen shot of a computer program

Description automatically generated

res4.py



Archivos generados hasta ahora

A screenshot of a computer

Description automatically generated

Lo que hace el módulo 2 es ensamblar todas las combinaciones posibles de variantes de subcódigo en el código base (CB), generando múltiples códigos fuente, y por ende, múltiples variantes de ejercicio (VE) según las combinaciones halladas, reemplazando las etiquetas #pycass sN en el código base por una variante específica del subcódigo N, y esta operación se repite para cada subcódigo existente.

En este ejemplo, se diseñaron cuatro variantes para el subcódigo 1, denotado como s1 en el código base (#pycass s1). Por lo tanto, se generaron cuatro códigos fuente (en este caso, estos son: res1.py, res2.py, res3.py y res4.py); cada uno de estos representa una variante de ejercicio.

Cabe indicar que es posible emplear múltiples subcódigos en el código base, la sangría del código base se tiene en cuenta en el reemplazo de etiquetas #pycass con variantes de subcódigo, y también es posible tener un número diferente de variantes por cada subcódigo en el código base.

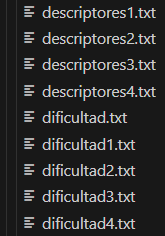
La idea detrás de estas acciones anteriores es obtener rápidamente, para todas las variantes de ejercicios consideradas, los códigos fuente que han sido propuestos en el diseño del ejercicio como solución a este último, con su respectivo código anexo para generar automáticamente datos de entrada aleatorios, fijos o secuenciales, es decir, sin la intervención de un ser humano, cada vez que se ejecuten esos códigos fuente, lo que permitiría la creación automática masiva de casos de prueba para todas las variantes de ejercicios en una etapa posterior, especialmente si los valores generados son aleatorios.

Módulo 3

A screenshot of a computer

Description automatically generated

Archivos generados por el módulo 3:



Archivos generados hasta ahora

A screenshot of a computer

Description automatically generated

descriptores1.txt

A black screen with white text

Description automatically generated

descriptores2.txt

A black screen with white text

Description automatically generated

descriptores3.txt

A black screen with white text

Description automatically generated

descriptores4.txt

A black screen with white text

Description automatically generated

dificultad.txt

A black screen with white text

Description automatically generated

dificultad1.txt

A black background with white text

Description automatically generated

dificultad2.txt



dificultad3.txt



dificultad4.txt

A black screen with white text

Description automatically generated

El módulo 3 genera archivos de descriptores y de niveles de dificultad para las variantes de ejercicio.

Los descriptores son cadenas de caracteres que son utilizadas para reemplazar partes de texto de la descripción del ejercicio. Cada variante de subcódigo debería tener asociado al menos un descriptor que identifica a esa variante.

En el ejercicio de ejemplo, se usaron dos descriptores para cada variante de subcódigo, porque la descripción del ejercicio se diseñó para que tuviese dos campos de texto que pueden cambiar dependiendo de la variante de ejercicio elegida. Estos campos de texto son denotados con la etiqueta <genai-desc N>, donde N es el número de descriptor, que puede empezar desde 1. En los archivos de texto dados, la cadena de caracteres de la primera línea corresponde al primer descriptor de la variante de ejercicio, y la segunda, al segundo. Entonces, dependiendo de la variante de ejercicio que se elija, el texto de la descripción del ejercicio es diferente cuando se reemplazan todas las ocurrencias de etiquetas con sus respectivos descriptores, según la variante de ejercicio que aplique.

descripcion.txt

<genai-desc 1> de dos números enteros:

Escriba un programa que lea dos números enteros y muestre en pantalla <genai-desc 2>.

La descripción de la primera variante de ejercicio sería así:

Suma de dos números enteros:

Escriba un programa que lea dos números enteros y muestre en pantalla su suma.

Por otro lado, el nivel de dificultad establecido a cada variante de ejercicio es importante porque los ejercicios mostrados a los estudiantes en el sistema de ejercicio y práctica adaptativa dependen del nivel de dificultad en el que se encuentra el estudiante durante su práctica.

Cabe indicar que es posible asignar un nivel de dificultad al código base y a cada una de las variantes de subcódigo.

Entonces, el nivel de dificultad de una variante de ejercicio (compuesto por un código base y un subconjunto de variantes de subcódigo, una variante por cada subcódigo en el código base) está establecido como la suma de los niveles de dificultad del código base y de las variantes de subcódigo que componen a la variante del ejercicio.

Por otro lado, también se puede realizar el cálculo del nivel de dificultad del ejercicio a partir del cual se establecen sus variantes como el promedio de los niveles de dificultad de todas las variantes de ejercicio generadas. Este cálculo podría ser útil si en el programa de ejercicios y práctica adaptativa se quisiere elegir primero de entre un grupo de ejercicios antes de determinar cuál es el siguiente ejercicio por mostrar al estudiante. Sin embargo, cabe advertir que este es un aspecto que, si bien fue explorado en el desarrollo de la solución propuesta al problema de investigación, su uso aún no tiene una justificación o sustento claro, pero esto se deja como idea al lector por si desea ahondar en ello más adelante.

El nivel de dificultad de un código base o de una variante de subcódigo puede ser mayor o igual a 0, pero es responsabilidad del docente asignar un nivel de dificultad adecuado para la variante de ejercicio que, pese a su nombre, puede determinarse no sólo según la dificultad del ejercicio, sino también, según su complejidad, y quizás, según su tiempo estimado de desarrollo.

Módulo 4

A blue and black rectangle with white text

Description automatically generated

Archivos generados por el módulo 4:



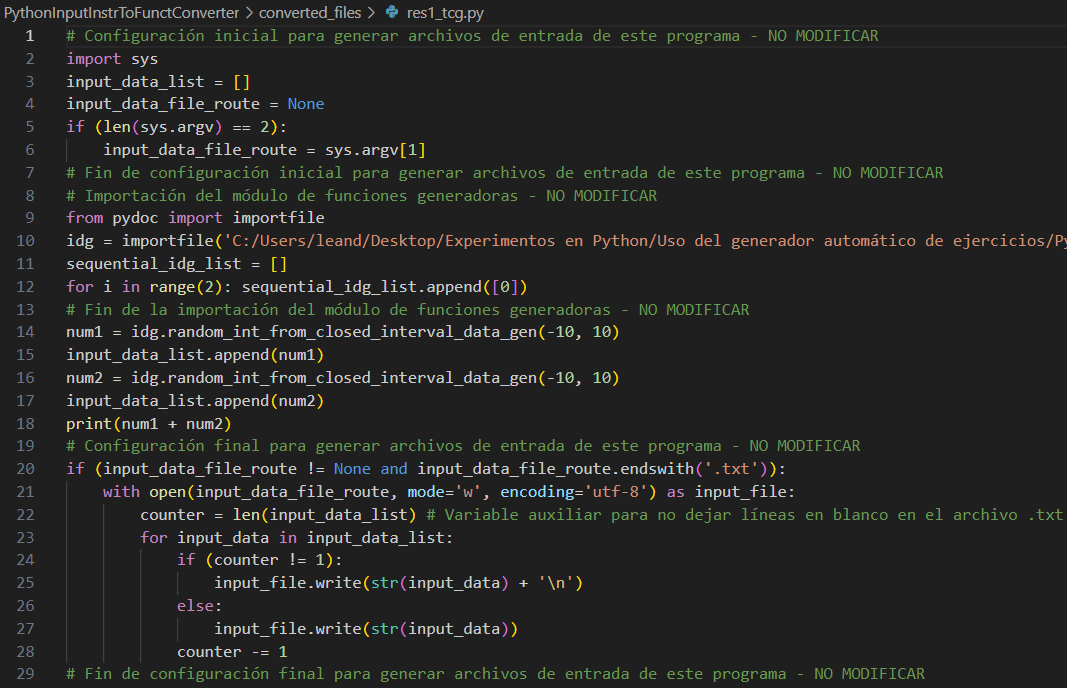






La palabra tcg significa *test case generator*; es decir, que estos son archivos específicamente diseñados para generar casos de prueba relacionados con cada una de las variantes de ejercicio.

res1\_tcg.py

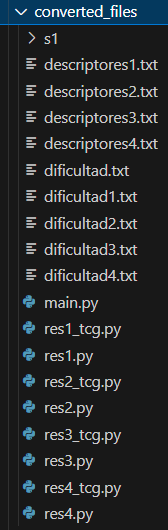


En este caso, los demás archivos generados, res2\_tcg.py, res3\_tcg.py y res4\_tcg.py, son similares a res1\_tcg.py, excepto por la impresión de datos determinada por una resta, multiplicación o una operación de elevar el primer número con el segundo, respectivamente, según lo establecido para cada variante de ejercicio. Si hubiere diferencias entre variantes de ejercicio sobre qué tipo de valores deberían ser generados como reemplazo de sus operaciones de lectura, entonces esas diferencias también estarían reflejadas en el archivo generado.

Sin embargo, lo que hace esencialmente el módulo 4 es añadir código al código fuente de cada una de las variantes de ejercicio, con el objetivo de brindar una funcionalidad adicional a este: crear un archivo .txt que contenga los datos de entrada generados por las funciones generadoras, línea por línea. El objetivo principal de esta creación de archivos .txt es precisamente permitir la generación automática de un archivo de datos de entrada (AE) para casos de prueba (CP), cada vez que se ejecuta el código fuente de la variante de ejercicio. Con respecto a la operación del módulo 4, cabe aclarar que este no crea casos de prueba, sino que sólo coloca algunos de los cimientos necesarios para ello, en pocas palabras. Por otro lado, es preciso mencionar que la automatización de la creación de casos de prueba para ejercicios de programación no sólo está fundamentada por el mecanismo de automatización de creación de archivos de datos de entrada que es posibilitada por el módulo 4, sino que también lo está por el trabajo realizado por el módulo 1, que reemplaza las instrucciones de lectura de los códigos fuentes asociados a las variantes de los ejercicios por funciones generadoras de valores que, al ser llamadas, producen un valor fijo, aleatorio, o uno determinado por un patrón, automatizando, de esta manera, el proceso de ingreso de datos, y finalmente, esta automatización de la generación de casos de prueba está asimismo soportada por las operaciones de salida que hacen parte de cada variante de ejercicio y fueron diseñadas en un principio por los docentes, antes de usar cualquiera de los módulos propuestos para la generación de ejercicios de programación. Otra de las características importantes que permite la inyección de código fuente que es realizada por el módulo 4 es que este permite que los códigos fuente asociados a variantes de ejercicio puedan ser ejecutados, por ejemplo, en una terminal o en una consola, con un argumento de línea de comando adicional: La cadena de caracteres correspondiente a la ruta del AE que será generado durante la ejecución, la cual también incluye el nombre del archivo a ser generado.

Lo anteriormente mencionado sobre la operación del módulo 4 implica, entonces, que si se ejecutase una vez el código fuente asociado a una variante de ejercicios, entonces no sólo se obtendría el resultado que debería producir el programa para resolver el ejercicio derivado de esa variante, sino que, al mismo tiempo, se produciría un archivo de entrada con los datos de entrada, generados por las funciones generadoras, que precisamente producen el resultado o salida producida por el programa durante la ejecución de esta. De la misma, si se realizan N ejecuciones, entonces se obtendrían N conjuntos de datos de entrada en N archivos distintos, y también se obtendrían N conjuntos de datos de salida como resultado de la ejecución del código fuente. Aun así, para automatizar completamente el proceso de creación de casos de prueba, todavía resta crear los archivos de datos de salida (AS) para los casos de prueba (CP) para obtener, así, las parejas de archivos de entrada/salida que requieren los casos de prueba para poder funcionar adecuadamente, pero esto se puede lograr con la ejecución del módulo 5.

Archivos generados hasta ahora:



Módulo 5



Directorios y archivos generados por el módulo 5:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer program

Description automatically generated 

AE y AS de CP de ejemplo que son generados para la primera variante de ejercicio (suma de dos números enteros):

A black screen with white text

Description automatically generated A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated A black screen with white text

Description automatically generated

(…)

AE y AS de CP de ejemplo que son generados para la segunda variante de ejercicio (resta de dos números enteros):

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

De esta forma también se generan los demás archivos de casos de prueba para las otras dos variantes de ejercicio (la multiplicación y la potenciación con dos números enteros).

Archivos de visibilidad.txt para cada una de las variantes de ejercicio:

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated





En este caso, esto significa que para todas las variantes de ejercicio, sólo el primer caso de prueba para cada una de las variantes de ejercicio (i.e., los datos descritos en los archivos entrada1.txt y salida1.txt) es visible a los estudiantes cuando ellos lo vean en el sistema de ejercicios y práctica adaptativa.

En definitiva, el módulo 5 genera, por cada variante de ejercicio, un directorio de casos de prueba que contiene tanto los AE y AS de los CP de la VE, además de un archivo adicional que permite configurar la visibilidad de los casos de prueba a los estudiantes; es decir, qué información de datos de entrada y de salida es visible al estudiante durante su práctica en el SEPPI.

Como puede observarse, en este caso se han configurado 10 CP para cada VE. Cada CP está compuesto por un par de archivos: entradaN.txt y salidaN.txt, donde N es el número del caso de prueba con respecto a un ejercicio o una variante de ejercicio y el primer caso de prueba es representado por el número 1.

Además, se puede ver también que el módulo 5 es el que genera directamente los AE y AS para CP, a partir de los códigos fuente de las variantes de ejercicios, los cuales, como se ha visto en los módulos 1 y 4, han tenido bastantes añadiduras.

Con respecto a cómo es posible la generación de los AE y AS de CP, se puede afirmar que la redirección de los canales o de los flujos de datos estándar es una manera recomendable de realizarlo.

Normalmente, el ingreso de datos a un programa se realiza por medio del teclado, ingresando caracteres –que pueden tener una representación visible o invisible en pantalla–, mientras que la salida de datos o el resultado producido por esta se visualiza en la pantalla que ve un usuario humano, que usualmente es el mismo que ingresa los datos al programa. Técnicamente, el ingreso y la salida de datos se realizan usualmente, por defecto, mediante dos canales o flujos de datos estándar: la entrada de datos estándar (*standard input stream, stdin*) y la salida de datos estándar (*standard output stream, stdout*), respectivamente. Esta es una forma básica de interacción humano-computador que es aceptable en el marco de los programas CLI, es decir, de las interfaces de líneas de comandos o de los programas que tienen una interfaz basada en texto en vez de una interfaz gráfica y que también pueden ser reconocidas como programas de consola o de terminal. Sin embargo, este escenario no es favorable para la automatización de la generación de casos de prueba, en donde se requiere una creación automática de archivos cuyo contenido depende tanto de los datos de entrada que deberían ingresarse a los códigos fuente propuestos para resolver ejercicios de programación como de los datos de salida que son producidos por estos mismos al ser ejecutados.

Entonces, para automatizar la creación de casos de prueba, se debe redireccionar el flujo de datos que es utilizado para ingresar datos al programa, y asimismo redireccionar el flujo de datos que es empleado para mostrar los resultados producidos por el programa, de tal manera que el ingreso de datos a un programa se realice con base en el contenido escrito en un archivo, y que asimismo la salida de datos del programa se describa en un archivo. De esta forma, es posible almacenar los datos de salida de cualquier programa, los cuales no se podían guardar anteriormente, sino que sólo se mostraban en pantalla durante la ejecución de esta.

Además de lo anterior, es importante considerar que esta manipulación de flujo de datos se puede realizar para la ejecución de cualquier programa en una terminal de un sistema operativo, incluyendo la ejecución de ambientes de ejecución, intérpretes o compiladores que a su vez ejecutan un programa a partir de un código fuente que puedan manejar.

En adición, cualquier programa puede ser invocado, llamado o ejecutado durante la ejecución de otro programa. Esto implica, por supuesto, que es posible ejecutar programas escritos en un lenguaje de programación mediante intérpretes, compiladores o ambientes de ejecución que pueden incluso ser llamados o invocados como un subproceso o un proceso hijo que es producido durante la ejecución de otro programa que se podría denominar proceso padre o programa principal.

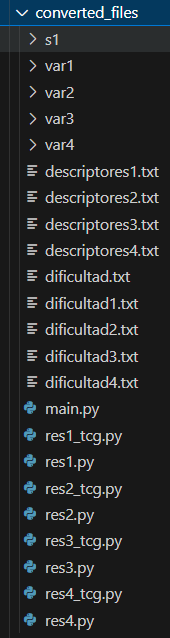
Por consiguiente, retomando el ejemplo del uso del sistema semiautomático de generación de ejercicios con el caso de las cuatro variantes a partir del ejercicio de sumar dos números enteros e imprimir su resultado en pantalla, se puede afirmar lo siguiente: si se cuenta con el módulo 5, un código fuente asociado a la variante de sumar dos números enteros, por ejemplo, y si se ejecuta este código durante la ejecución del módulo 5 por el número de veces equivalente a los casos de prueba que se quieren producir para el ejercicio, produciendo así un subproceso para cada ejecución del código en donde no se modifica el flujo *stdin* pero sí su flujo *stdout* a un archivo y en donde además se envía la ubicación junto al nombre y la extensión del archivo de datos de entrada a generar como un solo argumento de línea de comandos adicional, teniendo en cuenta que el nombre y ubicación del archivo difiere para cada ejercicio y para cada caso de prueba en particular a generar, entonces se pueden crear, por ejemplo, 10 casos de prueba de forma automática, con sus archivos de entrada y salida de datos, para la variante de ejercicio de sumar dos números enteros, y si esto mismo se realiza durante la ejecución del módulo 5 para las demás variantes de ejercicio (resta, multiplicación y potencia entre dos números enteros), entonces se pueden producir, de esta forma, 40 casos de prueba (si los valores producidos por las funciones generadoras son aleatorios), en una sola ejecución. Idealmente, los datos de entrada considerados para un solo ejercicio no deberían duplicarse.

La razón por la cual no se modifica el flujo de entrada de datos estándar en el caso anterior es porque en este esquema no se sabe de antemano cuáles van a ser los valores a ingresar al programa, sino que esto sólo se conoce cuando la función generadora determine el valor que será ingresado al programa – es decir, que los datos de entrada son producidos automáticamente dentro de la ejecución del código fuente y no afuera, o antes de su ejecución –; al respecto, no se debe olvidar que el valor que puede producir una función generadora puede ser aleatorio.

En contraste, una posible alternativa de implementación que quizás se podría considerar para ameritar una modificación al flujo de entrada de datos estándar es que los datos de entrada sean determinados de antemano, aleatoriamente o de forma determinística, en archivos de entrada de datos para casos de prueba, antes de que se ejecute el código fuente asociado a la variante de ejercicio, para que luego, al ejecutarse el código fuente de la variante, se produzcan los archivos de salida de los casos de prueba correspondientes a los datos de entrada dados por los archivos de entrada de datos. En este caso, no es necesario que el código fuente albergue las funciones generadoras de valores, pero este mecanismo sí debería ser incluido anteriormente de alguna manera, para poder producir automáticamente los datos de entrada.

En el módulo 5, básicamente, el usuario debe ingresar cuántos casos de prueba desea generar, y esto aplica para todas las variantes de ejercicio, suponiendo que todas las variantes de ejercicio deberían tener el mismo número de casos de prueba, aunque esto no es necesariamente correcto. Además, existen algunas opciones en el módulo que pueden brindar flexibilidad de uso. Por ejemplo, el docente puede indicar a partir de qué número de caso de prueba se quiere realizar la generación, en vez de la opción por defecto que es 1 (aplicado formalmente al primer caso de prueba), además de si quiere indicar que el primer caso de prueba generado sea visible o no (contar con por lo menos un caso de prueba es recomendado, puesto que es una buena práctica tener por lo menos un caso de prueba visible para los estudiantes como referencia para resolver el ejercicio), cuántos casos de prueba visible (numerados aleatoriamente) desea generar, incluyendo el caso de prueba visible indicado anteriormente (si así lo eligió), y si desea sobrescribir los archivos de casos de prueba visibles para las variantes de ejercicio.

Archivos generados hasta ahora:

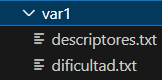


Módulo 6

A blue and black sign with white text

Description automatically generated

Archivos generados por el módulo 6:

 A screenshot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated



A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated A black screen with white text

Description automatically generated

Archivos generados hasta ahora:

A screenshot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

El módulo 6, pese a que su nombre indicaría que este genera nuevos archivos, no hace esto, sino que simplemente copia los archivos de descriptores y de nivel de dificultad asociados a cada variante de ejercicio y los ubica en su directorio correspondiente, en donde también se encuentran los archivos de entrada/salida de CP y de visibilidad de CP. Al respecto, cabe recordar que los archivos de descriptores son los que contienen las cadenas de caracteres que reemplazan a los espacios de texto reservados para ello en el archivo de descripción del ejercicio. Así, el texto de la descripción genérica del ejercicio es completada con base en la variante de ejercicio que sea seleccionada para ello. Por otro lado, los archivos de niveles de dificultad son los que informan al sistema de ejercicios y práctica adaptativa del nivel de dificultad de las variantes de ejercicio.

Con respecto al nombre dado al código fuente del módulo 6, la generación de ejercicios, en este caso, no debería entenderse como la creación de nuevos archivos, como se ha indicado antes. En cambio, se debería entender como el módulo que permite la culminación del proceso de creación de nuevos ejercicios o, dicho de otro modo, de la generación de nuevos ejercicios. En concreto, se termina de crear nuevos ejercicios en el sistema semiautomático de generación de ejercicios cuando se logra obtener un directorio de variantes de ejercicios que contenga todos los archivos necesarios para permitir la evaluación automática de estos en el SEPPI. Esto, en efecto, se logra con la ejecución del módulo 6. No obstante, es preciso mencionar que, de los archivos generados por los seis módulos, existen algunos de ellos que no son necesarios en SEPPI, y por ende, no requieren ser desplegados o trasladados a ese sistema. En cambio, por cada agrupación de variantes de ejercicios, sólo se requiere el contenido de los directorios específicos a estas variantes (e.g., var1, var2, var3, var4) y el archivo de texto que indica la media aritmética o el promedio del nivel de dificultad de las variantes de ejercicios producidas en esa agrupación, a partir de un solo ejercicio o código base. Alternativamente, podría ser posible ignorar el archivo del nivel de dificultad promedio si se producen variantes de ejercicio, pero si sólo se crea un solo ejercicio a partir de un código base sin variantes de subcódigo, entonces es conveniente, en ese caso, considerar el uso del archivo del nivel de dificultad promedio, que en ese caso específico corresponde al nivel de dificultad asignado al código base o al ejercicio base.

Archivos de ejercicios requeridos por SEPPI



A screenshot of a computer

Description automatically generated



Aspectos adicionales que deberían ser incluidos en el documento de tesis:

0a. Estructurar las subsecciones de la sección 5.3. (7)

0b. Explicar el proceso de diseño de los ejercicios de programación que debería emplearse en los sistemas propuestos. (6)

1. Hacer explícito en el documento el hecho de que el usuario puede elegir las funciones generadoras. (3) Esto se puede trabajar junto con el punto 2.

2. Explicar brevemente las funciones generadoras que se pueden realizar (Tabla) (4)

3. Diagrama de cómo se convierten las instrucciones de lectura en funciones generadoras (Tener en cuenta casos donde hay sangría o múltiples subcódigos) (1)

4. Diagrama de cómo se generan los casos de prueba (2)

5. Explicación referente al último módulo, el módulo 6, y la estructura de archivos que al final es obtenida para ser trasladada al otro sistema, el de ejercicio y práctica adaptativa. (Hecho)

6. Nota pequeña sobre un caso particular en el que se pueden hacer cambios manuales al código fuente de la variante de ejercicio para generar casos de prueba. (5) – Un buen ejemplo para ello es el Ejercicio 5. Otro ejemplo es el caso en que se requieren generar datos de prueba tanto predeterminados (o fijos) como aleatorios. En ese caso, se puede utilizar por lo menos dos veces la herramienta: uno para realizar los casos de prueba aleatorios, y otro, para generar los casos fijos. Alternativamente, el usuario también puede crear o modificar sus propios archivos de datos de entrada para casos específicos si así lo desea.

\*7. Revisar el cálculo del nivel de dificultad de los ejercicios generados.

\*8. Revisar la manera en que se calculan las combinaciones de elementos en una lista.

Aspectos opcionales que se podrían incluir en el documento de tesis:

1. Ubicar algunas consideraciones de detalles de funcionamiento encontradas en el documento. - Ya están escritas. - Si lo amerita.

2. Diagrama de cómo se generan las descripciones de los ejercicios - Quizás no.

Información irrelevante, pero correcta:

Teniendo en cuenta que el sistema de generación semiautomática de ejercicios está compuesto de varios módulos que funcionan como aplicaciones CLI, es decir, programas con una interfaz basada en texto en vez de una interfaz gráfica, aunque a estos también se les puede denominar programas de consola o de terminal.

Es posible manipular los canales o flujo de datos usados por defecto para ingresar datos a los módulos.

Estructura de la sección 5.3

Estructura original:

**5.3. Diseño del sistema semiautomático de generación de ejercicios con variantes a partir de plantillas de código**

**5.3.1. Configuración del directorio inicial de archivos de ejercicios**

**5.3.2. Conversión de instrucciones de lectura (IL) a funciones generadoras de datos (FG)**

**5.3.2.1. Instrucciones de lectura en Python**

**5.3.2.2. Funciones generadoras de datos**

**5.3.3. Ensamblador de código base (CB) y de variantes de subcódigo (VS) en códigos fuente asociados a ejercicios**

**5.3.4. Definición de niveles de dificultad y descriptores para los ejercicios**

5.2. Definición de niveles de dificultad para ejercicios de programación introductoria.

**5.3.5. Producción de archivos de entrada (AE) y de salida (AS) para los casos de prueba del ejercicio (CP)**

5.3.5.1. Añadidura de código fuente adicional al código base (CB) o variante de ejercicio (VE) previo a la generación de casos de prueba (CP) del ejercicio

**5.3.6. Generador de archivos adicionales necesarios para su uso en el sistema de ejercicios y práctica**

**5.3.7. Configuración del directorio final de archivos de ejercicios**

Nueva estructura:

**5.3. Diseño del sistema semiautomático de generación de ejercicios con variantes a partir de plantillas de código**

5.3.1. Diseño de los ejercicios de programación

5.3.2. Configuración del directorio inicial de archivos de ejercicios de programación

5.3.3. Conversión de instrucciones de lectura (IL) a funciones generadoras (FG) de datos de entrada

5.3.3.1. Conversión de IL a FG

5.3.3.2. Instrucciones de lectura en Python

5.3.3.3. Funciones generadoras de datos de entrada

5.3.4. Ensamblador de código base (CB) y variantes de subcódigo (VS) en códigos fuente de variantes de ejercicio (VE)

5.3.5. Definición de niveles de dificultad y descriptores para ejercicios de la programación introductoria

5.3.5.1. Cálculo del nivel de dificultad de los ejercicios de programación

5.3.5.2. Descripciones y descriptores de ejercicio

5.3.6. Generación de archivos de entrada (AE) y de salida (AS) de casos de prueba (CP) para evaluar automáticamente el ejercicio

5.3.7. Directorio resultante de ejercicios de programación generado como artefacto del sistema semiautomático de generación de ejercicios

Funciones generadoras:

a) Generadores de 1 valor fijo para todos los casos de prueba.

b) Generadores de 1 valor aleatorio, seleccionado de una forma repetible a partir de una lista dada por el usuario. Siempre retornará un valor aleatorio, independientemente de si se termina repitiendo una selección o no.

c)

d) Generadores de 1 valor fijo no aleatorio, seleccionado de una forma repetible de una lista dada por el usuario, siguiendo un orden de selección desde el elemento del índice 0 hasta el del índice n, para n casos de prueba. Si la lista se agota, y se vuelve a llamar a esta función desde la misma instancia de clase, entonces se retornará nuevamente el elemento del índice 0.

e) Función generadora compleja que acepta 1 o más funciones generadoras simples (con sus respectivos parámetros), las cuales son usadas 1 o más veces antes de usar la siguiente función generadora simple. Cuando todas las funciones generadoras simples se han ejecutado las veces que fueron especificadas, entonces se volverá a usar la primera función generadora simple, y se usará las mismas veces que fueron establecidas y aplicadas antes.

Otras menciones de funciones generadoras…

Cálculo del nivel de dificultad de los ejercicios:

# Considerando las siguientes fórmulas:

# Si el ejercicio tiene variantes:

# DE = PROMEDIO(DVE) = SUMA(DVE) / CANTIDAD(VE)

# DVE = SUMA(DVS) + DCB

# Si el ejercicio NO tiene variantes:

# DE = DCB

# Donde:

# DE = El nivel de dificultad de un ejercicio

# DVE = El nivel de dificultad de una variante de ejercicio

# DVS = El nivel de dificultad de una variante de subcódigo

# DCB = El nivel de dificultad del código base

# Paso 7: Pregunte al usuario por:

# a) El nivel de dificultad del código base del ejercicio: DCB

# b) El nivel de dificultad de cada una de las variantes de subcódigo del ejercicio: DVS

# c) El número de descriptores a usar por cada VS

# d) Cada uno de los descriptores a usar por cada VS

# En donde:

# a) Los niveles de dificultad tanto para el código base como para las variantes de subcódigo, y el número de descriptores, pueden ser mayor o igual que 0.

# b) Un descriptor puede ser, incluso, una línea vacía. => OK

Cálculo de las combinaciones de elementos en una lista:

Las combinaciones se generan seleccionando desde el último elemento hasta el primero.

s1, s2, s3, s4 = [2, 2, 3, 2]

1 : [1, 1, 1, 1] : s1:v1, s2:v1, s3:v1, s4:v1

2 : [1, 1, 1, 2] : s1:v1, s2:v1, s3:v1, s4:v2

3 : [1, 1, 2, 1] : s1:v1, s2:v1, s3:v2, s4:v1

4 : [1, 1, 2, 2] : s1:v1, s2:v1, s3:v2, s4:v2

5 : [1, 1, 3, 1] : s1:v1, s2:v1, s3:v3, s4:v1

6 : [1, 1, 3, 2] : s1:v1, s2:v1, s3:v3, s4:v2

7 : [1, 2, 1, 1] : s1:v1, s2:v2, s3:v1, s4:v1

8 : [1, 2, 1, 2]

9 : [1, 2, 2, 1]

10 : [1, 2, 2, 2]

11 : [1, 2, 3, 1]

12 : [1, 2, 3, 2]

13 : [2, 1, 1, 1]

14 : [2, 1, 1, 2]

15 : [2, 1, 2, 1]

16 : [2, 1, 2, 2]

17 : [2, 1, 3, 1]

18 : [2, 1, 3, 2]

19 : [2, 2, 1, 1]

20 : [2, 2, 1, 2]

21 : [2, 2, 2, 1]

22 : [2, 2, 2, 2]

23 : [2, 2, 3, 1]

24 : [2, 2, 3, 2]

Figura AA. Conversión de instrucciones de lectura (IL) en funciones generadoras (FG), dentro del sistema de generación semiautomática de ejercicios de programación

A screenshot of a computer

Description automatically generated

Figura BB. Generación de casos de prueba (CP) en el sistema de generación semiautomática de ejercicios de programación

Funciones generadoras (FG)

|  |  |  |  |
| --- | --- | --- | --- |
| Categoría | Tipo | Definición | Genera |
| Valor fijo | str | fixed\_str\_data\_gen(value: str) | La cadena de caracteres dada en el argumento de la función |
| Valor fijo | int | fixed\_int\_data\_gen(value: int) | El número entero dado dado en el argumento de la función |
| Valor fijo | float | fixed\_float\_data\_gen(value: float) | El valor punto flotante (número real) dado en el argumento de la función |
| Valor fijo | bool | fixed\_true\_data\_gen() | El valor True |
| Valor fijo | bool | fixed\_false\_data\_gen() | El valor False |
| Valor aleatorio por lista | str | random\_str\_data\_from\_list\_gen  (possible\_input\_values\_list: list) | Una cadena de caracteres de la lista dada como argumento de la función, elegida de forma aleatoria |
| Valor aleatorio por lista | int | random\_int\_data\_from\_list\_gen  (possible\_input\_values\_list: list) | Un número entero de la lista dada como argumento de la función, elegida de forma aleatoria |
| Valor aleatorio por lista | float | random\_float\_data\_from\_list\_gen  (possible\_input\_values\_list: list) | Un valor punto flotante de la lista dada como argumento de la función, elegida de forma aleatoria |
| Valor aleatorio por intervalo cerrado | Int | random\_int\_from\_  closed\_interval\_data\_gen  (min\_value: int, max\_value: int) | Un número entero que se encuentra entre un valor mínimo y uno máximo, incluyendo a estos últimos |
| Valor aleatorio por rango | Int | random\_int\_from\_range\_data\_gen  (data\_range: range) | Un número entero que se encuentra dentro de un rango de valores definido por un valor mínimo inclusivo, un valor máximo no inclusivo, y un valor de incremento o paso |
| Valor aleatorio por intervalo cerrado | float | random\_float\_from\_  closed\_interval\_data\_gen  (min\_value: float, max\_value: float) | Un número de punto flotante que se encuentra entre un valor mínimo y uno máximo que puede incluir a estos últimos |
| Valor aleatorio por intervalo cerrado, con precisión fija | float | random\_float\_from\_  closed\_interval\_with\_  fixed\_precision\_data\_gen(min\_value: float, max\_value: float, precision: int) | Un número de punto flotante que se encuentra entre un valor mínimo y uno máximo que puede incluir a estos últimos, y que tiene un número fijo de puntos decimales como argumento de la función |
| Valor aleatorio | bool | random\_strict\_bool\_data\_gen() | El valor True o False |
| Patrón de valores | str | fixed\_sequential\_  str\_data\_from\_list\_gen  (possible\_input\_values\_list: list, fsidg\_index\_item: list) | Una cadena de caracteres de la lista. En el primer llamado de esta función se genera el valor ubicado en el primer índice; en el segundo llamado, el segundo valor; etc. Al terminarse la lista, se vuelve a generar el primer valor de esta. |
| Patrón de valores | int | fixed\_sequential\_  int\_data\_from\_list\_gen  (possible\_input\_values\_list: list, fsidg\_index\_item: list) | Un número entero de la lista. En el primer llamado de esta función se genera el valor ubicado en el primer índice; en el segundo llamado, el segundo valor; etc. Al terminarse la lista, se vuelve a generar el primer valor de esta. |
| Patrón de valores | float | fixed\_sequential\_  float\_data\_from\_list\_gen  (possible\_input\_values\_list: list, fsidg\_index\_item: list) | Un valor punto flotante de la lista. En el primer llamado de esta función se genera el valor ubicado en el primer índice; en el segundo llamado, el segundo valor; etc. Al terminarse la lista, se vuelve a generar el primer valor de esta. |
| Patrón de valores | bool | fixed\_sequential\_  strict\_bool\_data\_from\_list\_gen  (possible\_input\_values\_list: list, fsidg\_index\_item: list) | Un valor booleano de la lista. En el primer llamado de esta función se genera el valor ubicado en el primer índice; en el segundo llamado, el segundo valor; etc. Al terminarse la lista, se vuelve a generar el primer valor de esta. |

Tabla 1. Funciones generadoras que se pueden utilizar en el sistema de generación semiautomática de ejercicios para reemplazar instrucciones de lectura de código fuente en Python

(¿Qué necesita el usuario para usar el sistema?)

Proceso de diseño de los ejercicios de programación que debería emplearse en los sistemas propuestos

En primer lugar, el docente debería diseñar un solo ejercicio de programación. Este es el ejercicio base. Luego, a partir de este ejercicio, se puede revisar la creación de variantes de ejercicio a partir de ese ejercicio base. Para crear variantes de ejercicio, se debería tomar porciones del algoritmo o el código que resuelva el ejercicio base, y modificarlos. En este caso, cada porción que se tome del ejercicio base para modificarlo corresponde a un subcódigo, y es básicamente un espacio de código que se toma para reemplazarlo con otros códigos fuente. Cada espacio de código o subcódigo puede tener, entonces, múltiples variantes; de ahí el nombre de variantes de subcódigo (VS). Entonces, (…)

Se debe considerar cuidadosamente cuántos casos de prueba quiere crear, y estos deben aplicar para todas las variantes de ejercicio.