Tutoriales Interactivos – Creación de lecciones

Enrique Martín Martín[a] (emartinm@ucm.es)
Revisor: Salvador Tamarit[b] (stamarit@dsic.upv.es)

[a] *Dpto. de Sistemas Informáticos y Computación*
Fac. Informática, Universidad Complutense de Madrid
[b] *Dep. Sistemes Informàtics i Computació*
Universitat Politècnica de València

9 de marzo de 2017

Resumen

La herramienta *Tutoriales Interactivos* organiza su contenido en distintas carpetas dentro del *directorio de temas*. Estas carpetas corresponden a cada uno de los lenguajes de programación soportados, y contienen un fichero YAML por cada tema. A su vez, los temas están definidos como una secuencia de lecciones, que contiene distintos elementos. En este documento se explicará la organización de carpetas y el formato concreto de los ficheros de temas.

Índice

1. Organización de carpetas

Todo el contenido de la aplicación se almacena en la *carpeta de temas*, cuya localización debe establecer el usuario desde la ventana de configuración. Esta carpeta contendrá un directorio por cada lenguaje para el que se quiera proporcionar temas, aunque pueden existir varios directorios para versiones alternativas del mismo lenguaje de programación (por ejemplo *Python 2.x* y *Python 3.x*, o *OpenJDK 6* y *Oracle Java 8*). Para determinar cuál es el lenguaje de programación asociado a un directorio la herramienta *Tutoriales Interactivos* inspecciona su nombre, comprobando si contiene alguna subcadena. Actualmente se soportan 4 lenguajes de programación, y la comprobación de nombre se realiza en el siguiente orden:

1. **Python**: si el nombre del directorio contiene[1] la subcadena «**python**». Para cada uno de estos directorios, la ventana de configuración mostrará una entrada para establecer la ruta del intérprete `python`.

2. **Java**: si contienen la subcadena «**java**» en su nombre. Para cada uno de estos directorios, la ventana de configuración mostrará dos entradas: una para establecer la ruta del compilador `javac` y otra para establecer la ruta del entorno de ejecución `java`.

3. **C++**: si contienen la subcadena «**c++**». Para cada uno de estos directorios, la ventana de configuración mostrará una entrada para establecer la ruta del compilador *GNU C++* (`g++`) o la del fichero de entorno de *Visual Studio* (`vcvars*.bat`).

4. **C#**: si el nombre contiene la subcadena «**c sharp**». Para estos directorios aparecerá una entrada en la ventana de configuración para establecer la ruta del compilador *Mono* (`mcs`) o el fichero de entorno de *Visual Studio* (`vcvars*.bat`).

2. Formato de los temas

Cada directorio que aparece en la carpeta de temas puede contener varios temas. Estos temas se definirán en ficheros con formato YAML[2] con extensión **obligatoria** `.yml`. El formato YAML sirve para representar información como texto plano en diccionarios clave-valor, listas y tipos de datos básicos como booleanos, números, cadenas, etc. Recomendamos consultar la sección «2.1 Collections» de la documentación de YAML (`http://www.yaml.org/spec/1.2/spec.html#id2759963`) para conocer las distintas maneras de representar diccionarios (*mappings*) y listas (*sequences*), puesto que admiten representaciones en una línea o en varias y ambas formas son utilizadas en los ejemplos de este manual.

[1] No importa si las subcadenas aparecen en mayúsculas o minúsculas en el nombre del directorio.

[2] `http://www.yaml.org/spec/1.2/spec.html`

```yaml
Subject: 1
Title: Titulo del tema
Intro: Breve explicación del tema
Lessons:
    - Title: Explicaciones #Primera leccion
      Elements: #Lista de elementos de la leccion
          - Elem: Text #Elemento de tipo explicacion
            Content: | #Texto de varias lineas
              Las explicaciones sirven para mostrar información al alumno.

              Estas esplicaciones se pueden dividir en varios párrafos, y con
              Markdown se pueden incluir **negritas**, *italicas* y ‘código‘.
          - Elem: Text
            Content: |
                **Imagen desde internet**
                ![texto alternativo](http://URL/imagen.png)

                **Imagen desde el directorio de temas con ruta relativa**
                ![triangulo](file://img/triangulo.jpg)

                **Se pueden incluir enlaces en las explicaciones:**
                [Texto del enlace](https://www.ucm.es)
    - Title: Preguntas de tipo test #Segunda leccion
      Elements:
          - Elem: Options #Pregunta de varias opciones
            Content: Pregunta de 3 opciones y una correcta #Texto de la pregunta
            Hint: Esto es la pista general de la pregunta
            Solution: [1] #Opciones correctas, empezando en 1
            Multiple: no #Hay varias opcinones correctas?
            Options: # Lista de opciones para mostrar
                - OPCIÓN CORRECTA
                - Opción incorrecta
                - Opción incorrecta
    - Title: Preguntas de tipo código #Tercera leccion
      Elements:
          - Elem: Code #Pregunta de codigo
            Content: | #Texto de la pregunta, varias lineas
                Estas preguntas solicitan al usuario framentos de código que
                son insertados en una plantilla correctora, que es evaluada.
            Gaps: 2 #Numero de huecos a rellenar
            Prompt: ["Codigo hueco 1","Codigo hueco 2"] #Informacion de cada hueco
            Hint: Esto es la pista general.
            File: correctores/plantilla.py #Plantilla correctora con 'huecos'
```

Figura 1: Código YAML de un tema con tres lecciones.

Un ejemplo de fichero de tema se puede observar en la Figura 1, donde se han añadido comentarios de línea (que comienzan con `#`) para aclarar algunos apartados. El formato concreto es un diccionario YAML con las siguientes claves y valores:

- `Subject`: (OBLIGATORIO) **número** de tema. Sirve para ordenar los distintos temas de un lenguaje a la hora de mostrarlos.
- `Title`: (OBLIGATORIO) **cadena de texto** con el título del tema.
- `Intro`: (OBLIGATORIO) **cadena de texto** que explica brevemente el objetivo del tema. Este texto puede incluir código *Markdown* o incluso código HMTL, como veremos en la Sección 3.
- `Lessons`: (OBLIGATORIO) lista de **lecciones** dentro del tema.

Cada una de las lecciones del tema se representa a su vez como diccionarios con dos claves:

- `Title`: (OBLIGATORIO) **cadena de texto** con el título de la lección.
- `Elements`: (OBLIGATORIO) lista de **elementos** que conforman la lección. Existen tres tipos de elementos: **explicaciones**, **preguntas de varias opciones** y **preguntas de código**. Los distintos elementos se tratarán con detalle en la Sección 4.

El título de tema, su introducción y los títulos de temas son los que *Tutoriales Interactivos* utiliza para mostrar las pantallas de selección de tema y lecciones. En la Figura 2 se puede ver cómo se mostraría la pantalla de selección para elegir entre las tres lecciones del tema definido en la Figura 1.

3. Formatos de texto aceptados

Los distintos elementos de una lección permiten mostrar contenido al usuario. Este contenido puede ser texto plano, pero también se puede formatear y añadir elementos visuales utilizando Markdown[3] o HMTL[4]. La opción preferida es código Markdown por ser el más sencillo y ser suficientemente flexible.

3.1. Markdown

Markdown proporciona muchas características para dar formato a un texto. La herramienta *Tutoriales Interactivos* soporta como mínimo las siguientes:

- **Negrita, itálica y código integrado en la línea**:

```
**negrita**, *italica*, ‘codigo de una linea‘
```

[3] `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`
[4] `https://www.w3schools.com/html`

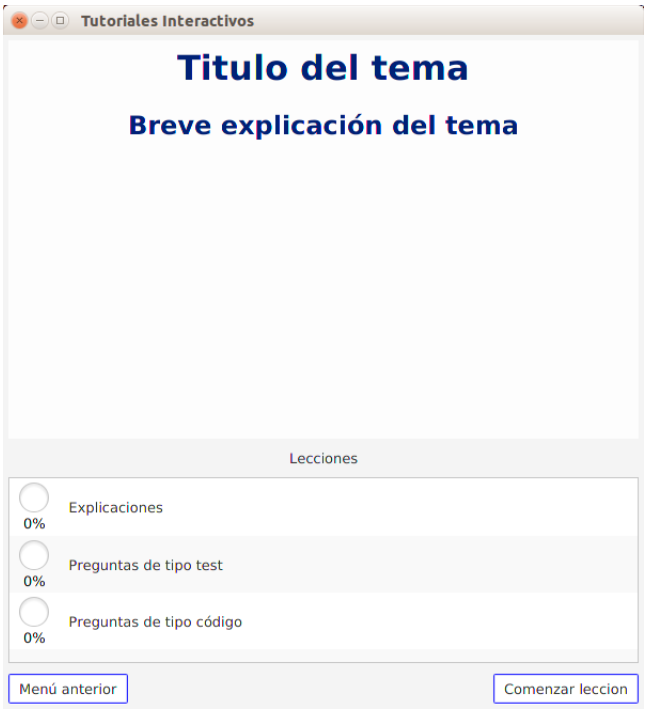

Figura 2: Pantalla de selección de lecciones

- **Bloques de código de varias lineas**:

```
‘‘‘
def f(n):
return n + 1
‘‘‘
```

- **Cabeceras de sección**:

```
# Cabecera 1
bla bla bla

## Cabecera 2
bla bla bla

### Cabecera 3
bla bla bla
```

- Listas numeradas y no numeradas:

```
* Elemento no numerado 1
1. elemento numerado anidado 1
1. elemento numerado anidado 2
1. elemento numerado anidada 3
* Elemento no numerado 2
* elemento no numerado anidado 1
* elemento no numerado anidado 2
```

- **Imágenes**: pueden ser imágenes remotas o almacenadas en el directorio del tema.

```
**Imagen remota**
![texto alternativo](https://URL/imagen.png)

**Imagen desde el directorio de temas, con ruta relativa**
![triangulo](file://img/triangulo.jpg)
```

  Todas las imágenes cuya ruta comience con `file://` serán consideradas imágenes locales cuya ruta es relativa al directorio donde reside el tema actual. Por ejemplo, si el tema actual reside en «`/opt/temas/Python 3.x`», la imagen «`file://img/triangulo.jpg`» se referirá al fichero situado en «`/opt/temas/Python 3.x/img/triangulo.jpg`».

- **Enlaces** que se abren en el navegador por defecto:

```
[Texto del enlace](https://www.ucm.es)
```

- **Notación matemática LaTeX**. Gracias el entorno MathJax[5] es posible incrustar código LaTeX en el texto. El código LaTeX debe encerrarse entre los símbolos `@@` para fórmulas en la misma línea y `@@@` para fórmulas en párrafos nuevos:

```
Consideremos el polinomio @@ax^2 + bx + c = 0@@
Cuando @@a \ne 0@@, existen dos soluciones:
@@@x = {-b \pm \sqrt{b^2-4ac} \over 2a}.@@@
```

- **Tablas**:

```
Cabecera|Cabecera|Cabecera
:-------|:------:|-------:   #Distinta alineacion horizontal
 1      | 2      | 3
 1      | 2      | 3
```

Recomendamos consultar el «Tema de prueba» que se puede encontrar en `https://github.com/emartinm/TutorialesInteractivos/blob/master/temas/Python%203.x/Tema0.yml` para ver en detalle las distintas características Markdown que se han presentado en esta sección.

[5] `https://www.mathjax.org/`

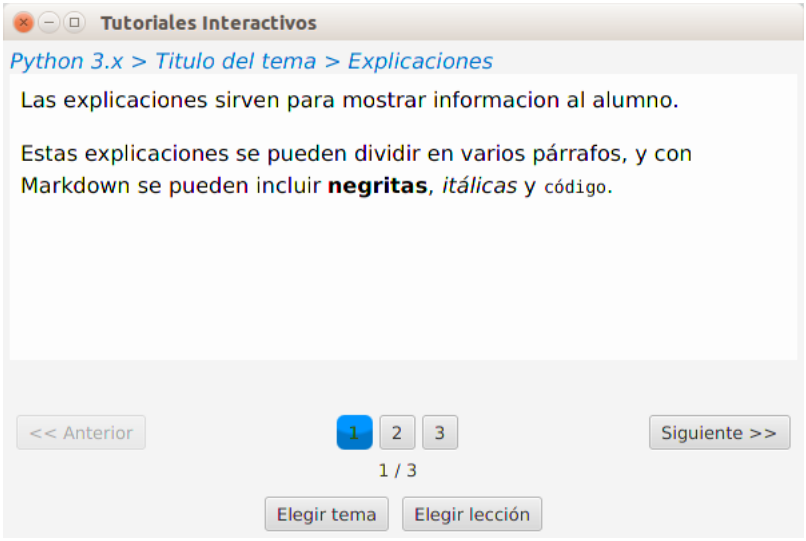

Figura 3: Pantalla mostrando la primera explicación de la lección «Explicaciones»

3.2. HMTL

En aquellas situaciones donde Markdown no es suficiente, se puede incluir código HMTL directamente. Esto puede servir para incrustar vídeos y otros elementos interactivos.[6] Por ejemplo el siguiente código incrustaría el vídeo `https://www.youtube.com/embed/PDpMgx7avzA` como un `iframe`:

```
<iframe width="640" height="360" src="https://www.youtube.com/
  embed/PDpMgx7avzA" frameborder="0" allowfullscreen target="
  _self"></iframe>
```

4. Formato de los elementos de una lección

4.1. Explicaciones

Los elementos de tipo *explicación* sirven para mostrar al alumno texto y otros contenidos gráficos. Se representan como un diccionario de dos claves:

- `Elem: Text` (OBLIGATORIO)

[6] La herramienta *Tutoriales Interactivos* utiliza internamente la componente *WebView* de JavaFX, que está basada en WebKit (`https://webkit.org/`). Por lo tanto, la capacidad para procesar HTML será algo menor al de los navegadores de escritorio usuales y habrá algunas características que no funcionen correctamente o directamente no estén soportadas.

- `Content`: (OBLIGATORIO) **cadena de texto** con el contenido que se quiere mostrar. Puede ser texto plano, código Markdowno HTML. En la Figura 1 se pueden ver ejemplos de contenidos Markdown en las líneas 9–12 y 15–22

4.2. Preguntas de varias opciones

Este tipo de elementos sirve para preguntas en las que el alumno debe elegir la opción u opciones correctas entre varias disponibles. En el fichero YAML las preguntas de varias opciones se representan como un diccionario con las siguientes claves y valores:

- `Elem: Options` (OBLIGATORIO)
- `Content:` (OBLIGATORIO) **cadena de texto** con el enunciado de la pregunta. Al igual que en las explicaciones, este texto puede contener código Markdown o HTML.
- `Hint:` (OPCIONAL) **cadena de texto** con una pista sobre la solución a la pregunta que el alumno puede visualizar pulsando el botón «Pistas».
- `Options:` (OBLIGATORIO) **lista de cadenas** con las distintas opciones que se mostrarán al alumno. Nótese que en YAML las listas se pueden representar en una línea con la notación `[e_1, e_2, ..., e_n]` o en varias líneas usando guiones (`-`) para cada elemento.
- `Multiple:` (OPCIONAL) **booleano** que indica cómo han de mostrarse las distintas opciones de la pregunta. En caso de elegir `no` las opciones se muestras mediante *radio buttons* y únicamente se permite elegir una opción, en caso de elegir `yes` las opciones se muestran mediante *checkboxes* y se pueden elegir una o más opciones. En caso de no incluir esta clave, el valor por defecto es `no`.
- `Solution:` (OBLIGATORIO) **lista de números** con las posiciones de las opciones correctas, donde la primera opción tiene posición `1`. Este campo puede contener la lista vacía (`[]`) si ninguna opción es correcta. Para superar una pregunta es necesario marcar exactamente las opciones que se indican en este campo.

Las lineas 25–33 de la Figura 1 contienen un ejemplo de pregunta de tres opciones donde únicamente la primera es correcta. A la hora de mostrar esta pregunta, *Tutoriales Interactivos* generaría la pantalla de la Figura 4.

4.3. Preguntas de código

Las preguntas de tipo código permiten solicitar al alumno uno o varios fragmentos de código y evaluar su resultado. Para dicha evaluación los fragmentos de código introducidos por el alumno se insertan en los *huecos* definidos en una *plantilla correctora* (ver más adelante), que está escrita en el mismo lenguaje

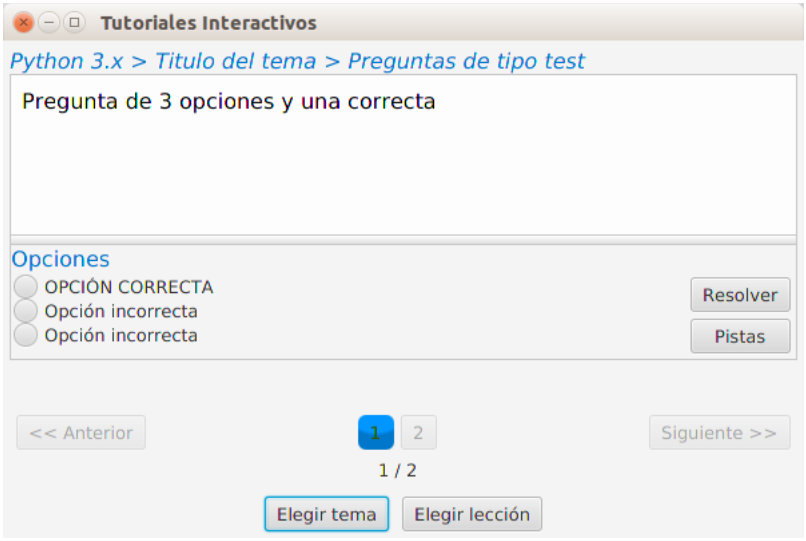

Figura 4: Pantalla mostrando una pregunta de tres opciones con solución única.

de programación que está aprendiendo el alumno. Esta plantilla se rellena y se ejecuta para obtener su veredicto, lo que involucra distintas fases dependiendo del lenguaje de programación: invocar directamente al intérprete (Python), compilar la plantilla rellena y luego invocar el binario generado (C++ y C#), compilar la plantilla rellena y luego interpretar el fichero generado (Java). La estructura de las plantillas correctoras se detallará en la Sección 4.3.1.

En el fichero YAML las preguntas de código se representan como un diccionario con las siguientes claves y valores:

- `Elem: Code` (OBLIGATORIO)
- `Content:` (OBLIGATORIO) **cadena de texto** con el enunciado de la pregunta. Como es usual, este texto puede contener código Markdown o HTML.
- `Gaps:` (OPCIONAL) **número entero** (> 0) que indica el número de huecos que debe rellenar el alumno. Si no se incluye este campo, el valor por defecto es 1.
- `Prompt:` (OPCIONAL) **lista de cadenas de texto** con tantos elementos como se indique en el campo `Gaps`. Indica el texto por defecto que debe aparecer en cada campo de texto cuando está vacío. Sirve para dar indicaciones al alumno sobre lo que debe introducir en cada hueco.
- `Hint:` (OPCIONAL) **cadena de texto** con una pista sobre la solución a la pregunta que el alumno puede visualizar pulsando el botón «Pistas».

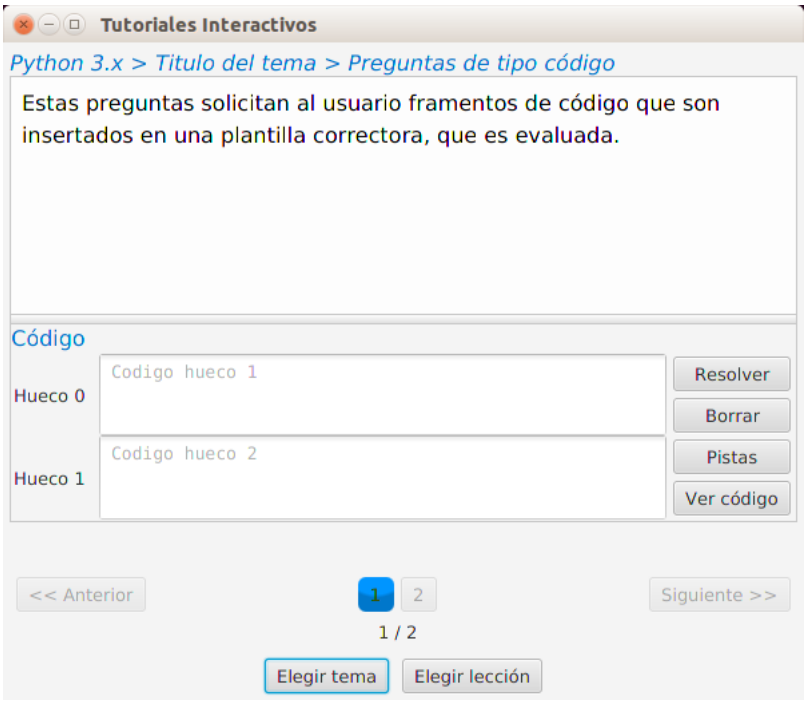

Figura 5: Pantalla mostrando una pregunta de código con dos huecos.

- `File:` (OBLIGATORIO) **cadena de texto** con la ruta de la plantilla correctora *relativa al directorio en el que está el tema actual*. Por ejemplo, si el tema actual está en el directorio «`/opt/temas/Python 3.x`» y la ruta de la plantilla correctar es «`correctores/tema1_p3.py`», la ruta completa de la plantilla será «`/opt/temas/Python 3.x/correctores/tema1_p3.py`».

Las líneas 36–43 contienen una pregunta de tipo código con dos huecos y que se corrige usando la plantilla. `correctores/plantilla.py`. A la hora de mostrar esta pregunta, *Tutoriales Interactivos* generaría la pantalla que se puede ver en la Figura 5.

4.3.1. Plantillas correctoras

Las plantillas correctoras son programas en los que se ha insertado uno o más *huecos*. Para indicar en qué punto de la plantilla hay un hueco, se utiliza el marcador `@@@CODE@@@`. En la plantilla correctora deben aparecer tantas marcas `@@@CODE@@@` como huecos se han definido en la clave `Gaps` del elemento. La Figura 6 muestra un ejemplo de plantilla de un solo hueco para Python, considerando una pregunta en la que se debe asignar el valor 4 a la variable `i`. Como se puede observar, existe un único hueco en la línea 6.

A la hora de corregir una pregunta de tipo código, la herramienta *Tutoriales*

```
# -*- coding: UTF-8 -*-
import sys
import json

def corrige(filename):
    @@@CODE@@@
    dicc = {}
    things = locals()
    if 'i' in things:
        if things['i'] == 4:
            dicc = {'isCorrect': True}
        else:
            dicc = {'isCorrect': False,
                    'typeError': "Valor incorrecto",
                    'Hints': ['La variable tiene valor' + str(i),
                              'Deberia tener valor 4.'] }
    else:
        dicc = {'isCorrect': False,
                'typeError':"Variable 'i' no asignada",
                'Hints':["Asigna algun valor a la variable 'i'"]}

    with open(filename, 'w') as outfile:
        json.dump(dicc, outfile) #Vuelca 'dicc' como JSON
        sys.exit(0);

if __name__ == "__main__":
    corrige(sys.argv[1]) #sys.argv[1] es la ruta del fichero JSON
```

Figura 6: Ejemplo de plantilla correctora en Python

Interactivos inserta en orden los fragmentos de código del alumno en los distintos huecos y ejecuta la plantilla. Para rellenar estos huecos se tiene en cuenta el *espaciado antes de la marca de hueco*. Volviendo al ejemplo, el hueco de la Figura 6 está a 3 espacios del inicio de fichero. Eso significa que a la hora de reemplazar el hueco con código del alumno, **se insertarán 3 espacios al inicio de cada línea**. Imaginemos que el código del alumno es:

```
a = 1
b = a * 4
i = b
```

A la hora de reemplazar el hueco en la plantilla esta quedaría como se ve en la Figura 7, donde todas las líneas han sido precedidas por exactamente 3 espacios.

Las plantillas correctoras no imponen ninguna restricción a la hora de dar nombre a sus distintas componentes: variables, métodos, etc. La única restric-

```
1  # -*- coding: UTF-8 -*-
2  import sys
3  import json
4
5  def corrige(filename):
6      a = 1
7      b = a * 4
8      i = b
9      dicc = {}
10     ...
```

Figura 7: Plantilla correctora rellena con código del alumno

ción general es que deben estar definidas en un solo fichero y que deben contener una función principal que acepte un parámetro, realice las pruebas necesarias y vuelque el veredicto en un fichero JSON (ver siguiente sección). El caso de **Java** tiene una **restricción adicional**: el método `main(String[] args)` debe estar definido en una clase con nombre `Corrector`, puesto que será la clase que usará *Tutoriales Interactivos* para lanzar la corrección. Recomendamos revisar los correctores de prueba que están en la carpeta `https://github.com/emartinm/TutorialesInteractivos/tree/master/temas` para ver ejemplos de correctores en los distintos lenguajes soportados.

4.3.2. Corrección de plantillas: JSON

Cuando se ejecuta una plantilla rellena, la herramienta *Tutoriales Interactivos* pasa como parámetro una ruta de fichero absoluta donde se debe almacenar el veredicto de la corrección en formato JSON[7]. Este fichero debe contener un diccionario con las siguientes claves:

- `'isCorrect'`: (OBLIGATORIO) **booleano** (`true`/`false`) que indica si el código es correcto.
- `'typeError'`: (OPCIONAL) **cadena de texto** con un mensaje sobre la naturaleza del error.
- `'Hints'`: (OPCIONAL) **lista de cadenas de texto** con pistas detalladas sobre lo que ha fallado o sobre cómo se podría arreglar el código enviado. Cada elemento de la lista se mostrará como una línea de texto cuando el alumno pulse el botón «Más pistas».

La Figura 6 muestra cómo se construyen 3 diccionarios para volcar el resultado en formato JSON: en la línea 11 el código es correcto, por lo que sólo se crea la clave `'isCorrect'` con valor `True`; por otro lado, en las líneas 13 y 18 la clave

[7] `http://www.json.org/`

`'isCorrect'` toma valor `False` y se añaden las otras dos claves opcionales con mensajes concretos para el alumno.

Como hemos comentado anteriormente, cuando *Tutoriales Interactivos* corrige una pregunta de código primero rellena la plantilla y luego la ejecuta. Si esta ejecución lanza algún error (por ejemplo errores de sintaxis durante la compilación, excepciones durante la ejecución...) el alumno recibirá un mensaje indicando esta situación. Además, si la ejecución tarda más de 2 segundos será abortada y se mostrará un mensaje al usuario indicando esta situación, para evitar posibles bucles infinitos. Únicamente en el caso de que la ejecución termine dentro del límite y tenga éxito, la herramienta leerá el fichero JSON generado y mostrará los resultados al alumno.

4.3.3. Visualización de plantillas rellenas

Por último, *Tutoriales Interactivos* permite que el alumno visualice cómo serán rellenados los huecos con su código sin llegar a ejecutarlo. Para ello es necesario que en la plantilla correctora se haya marcado qué fragmento de código se debe mostrar al alumno mediante las marcas `@@@SNIPPET@@@`. Consideremos una pregunta de ćodigo en la que hay que escribir el cuerpo de una función Python que duplica el argumento `n`. La plantilla correctora contendrá el siguiente código:

```
#@@@SNIPPET@@@
def duplica(n):
    @@@CODE@@@
#@@@SNIPPET@@@
```

Como se puede observar, toda la función `duplica` aparece encerrada entre marcas `@@@SNIPPET@@@`. Si el usuario rellena el código «`return n * 2`» en el campo de texto y pulsa el botón «Ver Código» la herramienta reemplazará el hueco y posteriormente mostrará las líneas que están entre las marcas `@@@SNIPPET@@@`. El resultado que vería el alumno sería:

```
def duplica(n):
    return n + 2
```

Obsérvese que las líneas donde aparecen las marcas nunca se mostrarán cuando el alumno visualice el código, sino únicamente las líneas que están **entre las marcas**. Normalmente las marcas `@@@SNIPPET@@@` aparecerán dentro de comentarios, para no afectar a la compilación/ejecución de la plantilla.

5. Crear una lección nueva

Para crear una lección nueva los pasos a seguir serían:

1. Crear un subdirectorio dentro de la carpeta de temas para el lenguaje de programación (o versión) concreto, si no existe ya.

2. Crear un fichero YAML con extensión `.yml` dentro de dicho subdirectorio. En lugar de escribirlo desde cero recomendamos tomar el «Tema de prueba» que se puede encontrar en `https://github.com/emartinm/TutorialesInteractivos/blob/master/temas/Python%203.x/Tema0.yml` y realizar modificaciones sobre él. El formato YAML es sensible al sangrado, así que es importante tenerlo en cuenta a la hora de anidar los distintos elementos. Es por ello que recomendamos utilizar el «Tema de pruebas» como punto de partida.

3. Para crear las plantillas correctoras, recomendamos tomar las usadas en las lecciones de prueba de cada uno de los lenguajes de programación (`https://github.com/emartinm/TutorialesInteractivos/tree/master/temas`) y realizar las modificaciones necesarias respetando el esqueleto general.