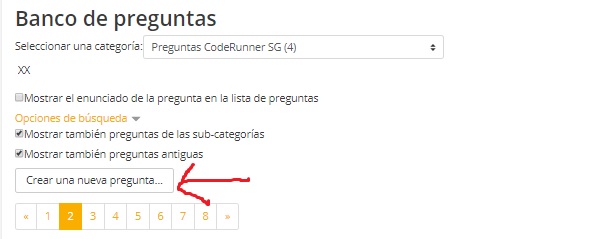
**Informe acerca del uso del CodeRunner.**

CodeRunner Es un complemento para moodle que permite crear quizes o cuestionarios, en el cual los estudiantes escriben respuestas en algún lenguaje de programación soportado y automáticamente el CodeRunner las prueba con algunos casos de prueba especificados, los cuales se le muestran al estudiante (o no) y después de hacer un envío, ellos obtienen una retroalimentación del resultado. CodeRunner soporta varios lenguajes de programación como: R, Python, C, Java, PHP entre otros.

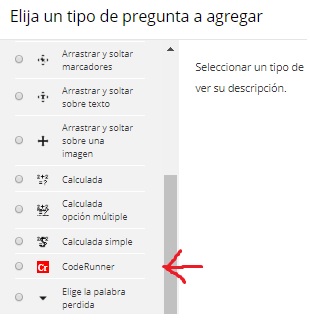
En un principio se explicará cómo funcionan las preguntas, como se pueden configurar y los distintos tipos de preguntas que se pueden formular.

**¿Cómo crear una pregunta para CodeRunner?**

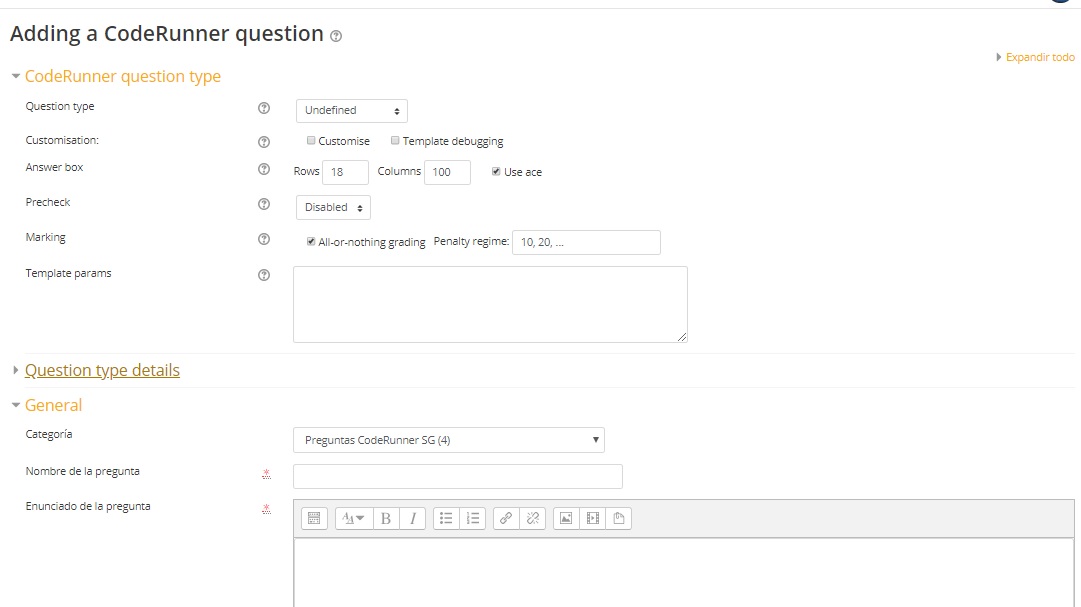
Estando dentro de la página del curso en el sistema, se puede dirigir hacia el banco de preguntas y allí seleccionar “Crear nueva pregunta”:



En la pantalla siguiente, se pide elegir un tipo de pregunta a agregar, allí seleccionamos la opción CodeRunner:



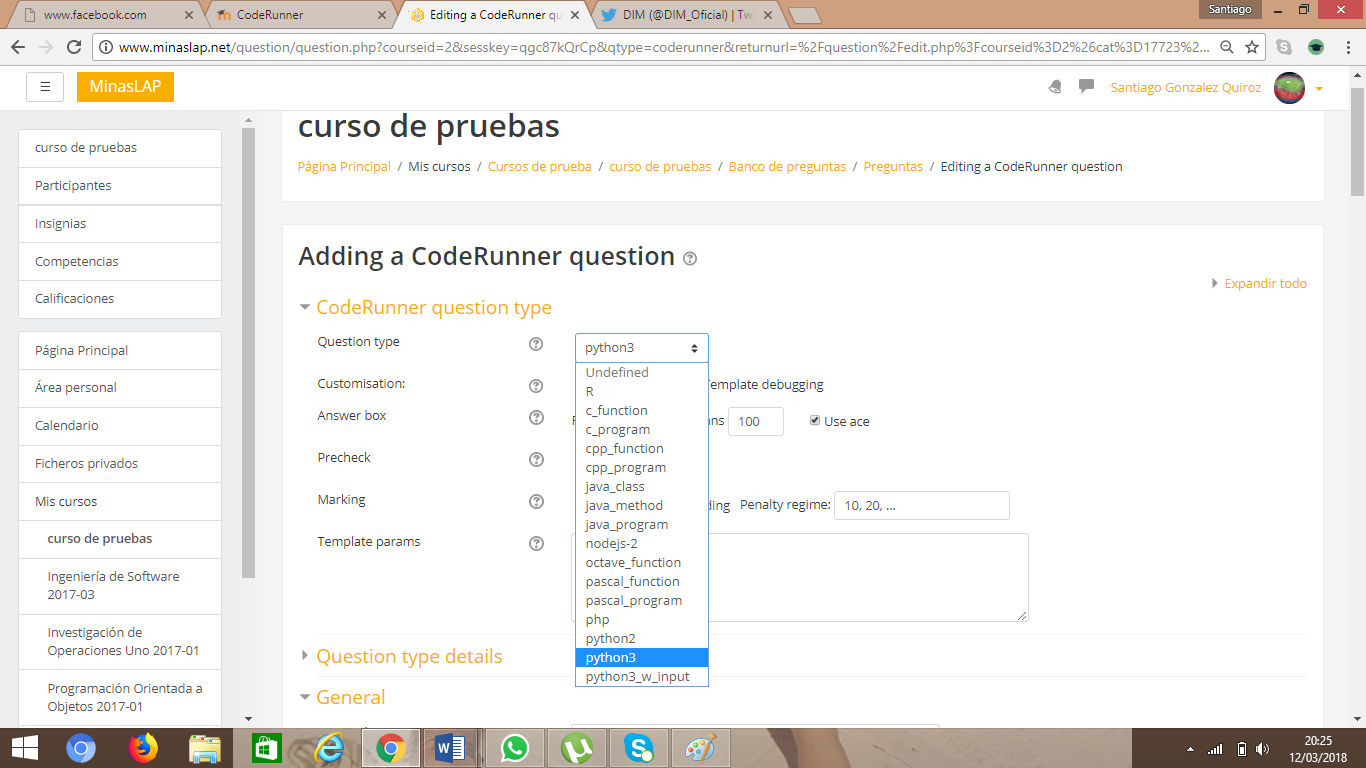
Le damos la opción “Agregar” y se despliega la pantalla de creación, personalización y edición de las preguntas de tipo CodeRunner:



Esta pantalla esa compuesta por muchas partes, las cuales se llamarán ahora “Parámetros de creación”, los parámetros más importantes se van a explicar a continuación:

**\*Los parámetros que no se expliquen o tengan en cuenta en este tutorial, es porque por ahora no son importantes o no han sido investigados a fondo.**

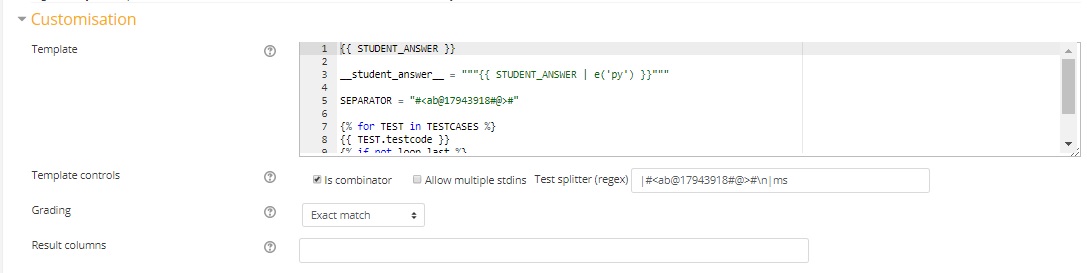
* ***Question type*:** Por medio de este parámetro se le aclara a la plataforma que tipo de pregunta y en que lenguaje de programación se trabajara. La plataforma soporta preguntas de tipo programa simple o de tipo función. Algunos lenguajes de programación requieren indicar especialmente que será una función, otros (como Python) no requieren esa especificación.



* ***Customisation*:**

****

*Customisation*es uno de los parámetros más importantes, ya que al seleccionar este, se puede configurar la forma de evaluar la pregunta, la plantilla para evaluarla y el tipo de calificación deseada. Si *customise* esta chequeado, más abajo aparecerá el desplegable “*Customisation*” que se ve así en la pantalla:



Es importante tener en cuenta dos cosas para entender la *Customisation*, la primera es la plantilla (*Template*) usada para evaluar un ejercicio y la segunda es el *grading* para calificar las respuestas. Primero hablemos de las plantillas para evaluar un ejercicio,

Existen dos plantillas predefinidas: “*Is combinator*” y *“per-test”*.

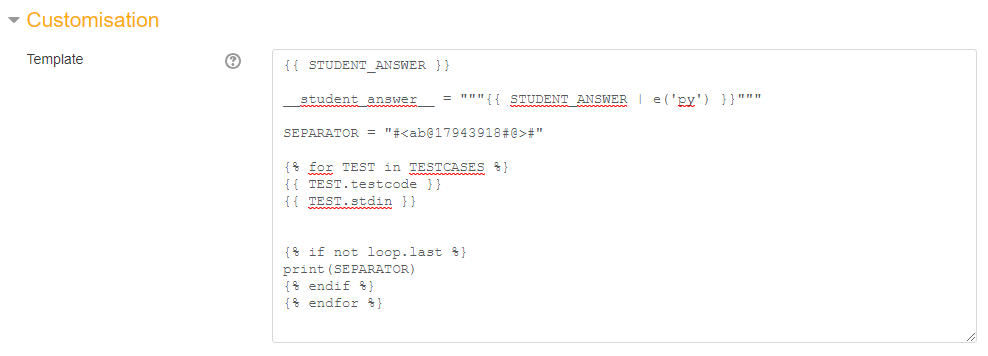
**\**Combinator*:** El combinator se encarga de realizar una sola prueba en la que se incluyen todos los casos de prueba definidos en un ejercicio, es decir, en una sola ejecución evalúa internamente todas las pruebas. El *combinator* es la mejor plantilla para trabajar por la forma en que evalúa los ejercicios. *Combinator* es la plantilla con la que por defecto trabaja CodeRunner en caso de que no se seleccione la opción “*Customisation*”.

**\**Per-test***: Con esta plantilla se ejecuta una prueba por cada caso de prueba definido. Es un método menos eficiente que el combinator y es menos usado. En este punto, es importante resaltar aspectos importantes sobre la plantilla:

**\***La plantilla construye un programa ejecutable, el cual es procesado por el motor twig de procesamiento de plantillas: <https://twig.symfony.com/>

El beneficio que se obtiene al hacer el uso de este motor de plantillas (escrito en PHP), es que permite mucha esteticidad en el código, optimización a la hora de la ejecución, además de ser integrable con varios entornos de desarrollo en PHP.

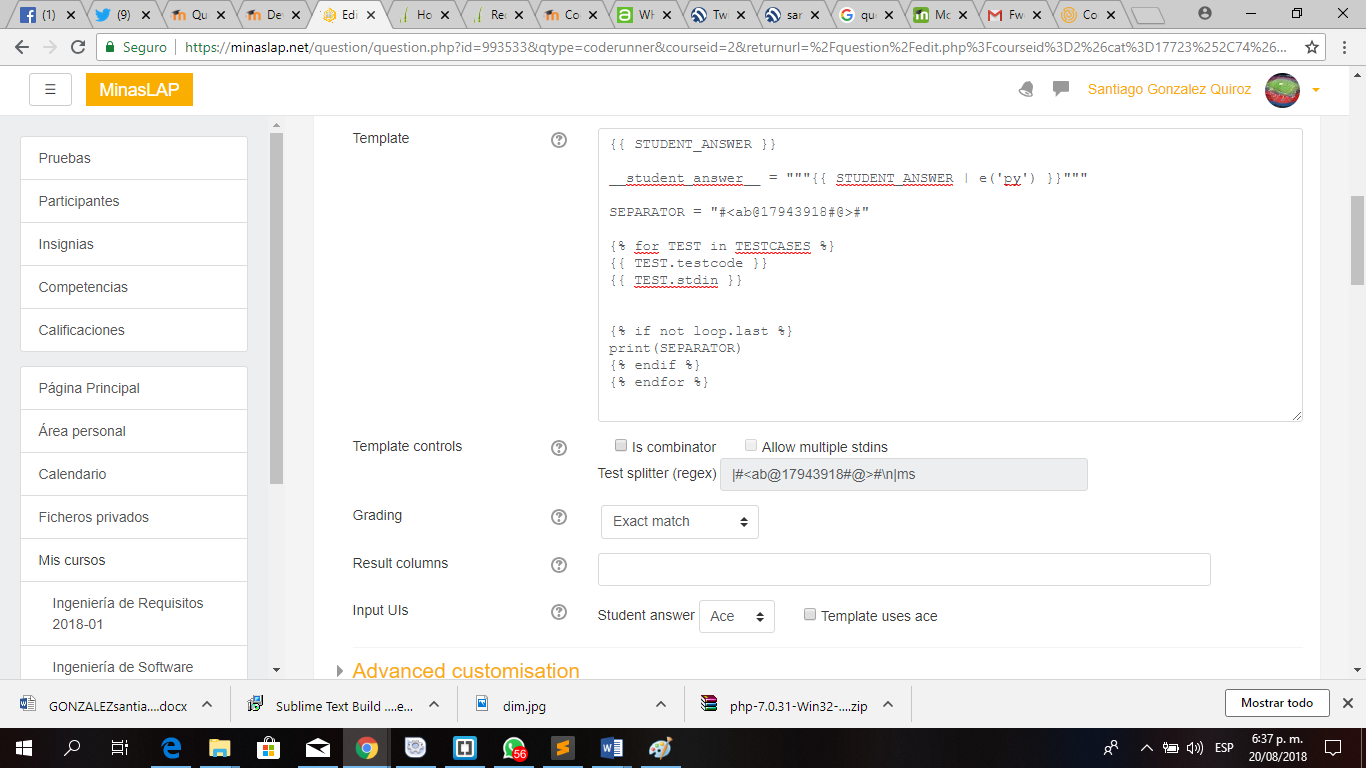
El programa que construye el motor Twig se puede ver en el apartado de customisation:



Twig usa expresiones que toman valores al momento de poner en marcha la plantilla. STUDENT\_ANSWER, TEST.testcode, son algunas de dichas expresiones. STUDENT\_ANSWER toma el valor de la respuesta que da el estudiante, TEST.testcode toma el valor del caso de prueba indicado, TEST.stdin toma el valor de los inputs indicados en el caso de prueba. El per-test, como se dijo antes, crea una prueba para cada caso de uso, eso explica el {% for TEST int TESTCASES %} por medio de un ciclo hace todas las pruebas.

Es importante reconocer que en Twig la estructura {{variable}} sirve para plasmar el valor de una variable dentro de la plantilla, mientras que la estructura {% código %} sirve para meter código de control: Ciclos, condicionales.

Para seleccionar la plantilla per-test, primero que todo se debe haber seleccionado la opción customise al principio de la declaración del ejercicio y en template controls se deschequea “Is\_combinator” (Que es la plantilla por defecto)

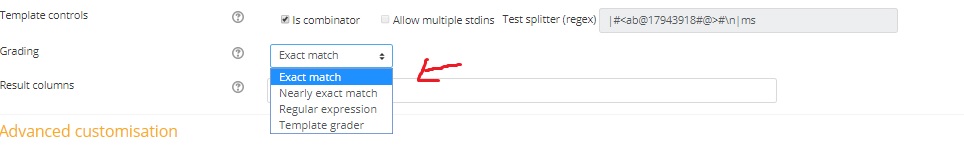


Entre una y otra plantilla no hay gran diferencia representativa, lo importante es que es más comodo y eficiente trabajar con la plantilla combinator.

**Para los cursos, se propone y se trabajara de acá para adelante con la plantilla “*Combinator*”.**

Ahora hablemos del *grading*, el cual es otro parámetro importante dentro de la pregunta:

**El *grading*** sirve para comparar el output esperado en cada caso de prueba con el output que produce la respuesta del estudiante. Se puede seleccionar como exactamente igual, cercanamente igual o regularmente igual (mala respuesta).



Simplemente se selecciona la forma en que se quiere que el grading actúe en la pregunta. A continuación, presentamos lo que cada tipo de grading permite:

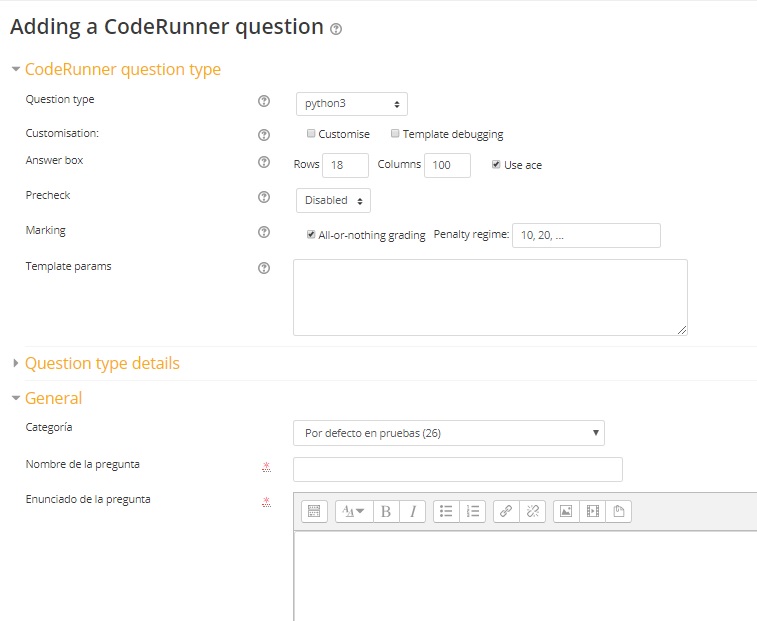
**\**Exact match*:** Espera que la respuesta del estudiante coincida exactamente con la salida especificada en el caso de prueba.

**\**Nearly exact match*:** es similar al anterior pero no distingue entre mayúsculas y minúsculas y tolera las variaciones en la cantidad de espacio en blanco

**\**Regular expression*:** Busca alguna leve similitud entre la respuesta y la salida del caso de prueba, es el grading más “Mediocre”.

**\*\*Vale aclarar que si no se quiere modificar el grading (Que por defecto viene en exact match) y asumiendo que se va a trabajar con la *“is combinator” template* (También viene por defecto), NO es necesario hacer uso de la opción *Customisation*, simplemente no se chequea y se trabaja con las configuraciones por defecto.**

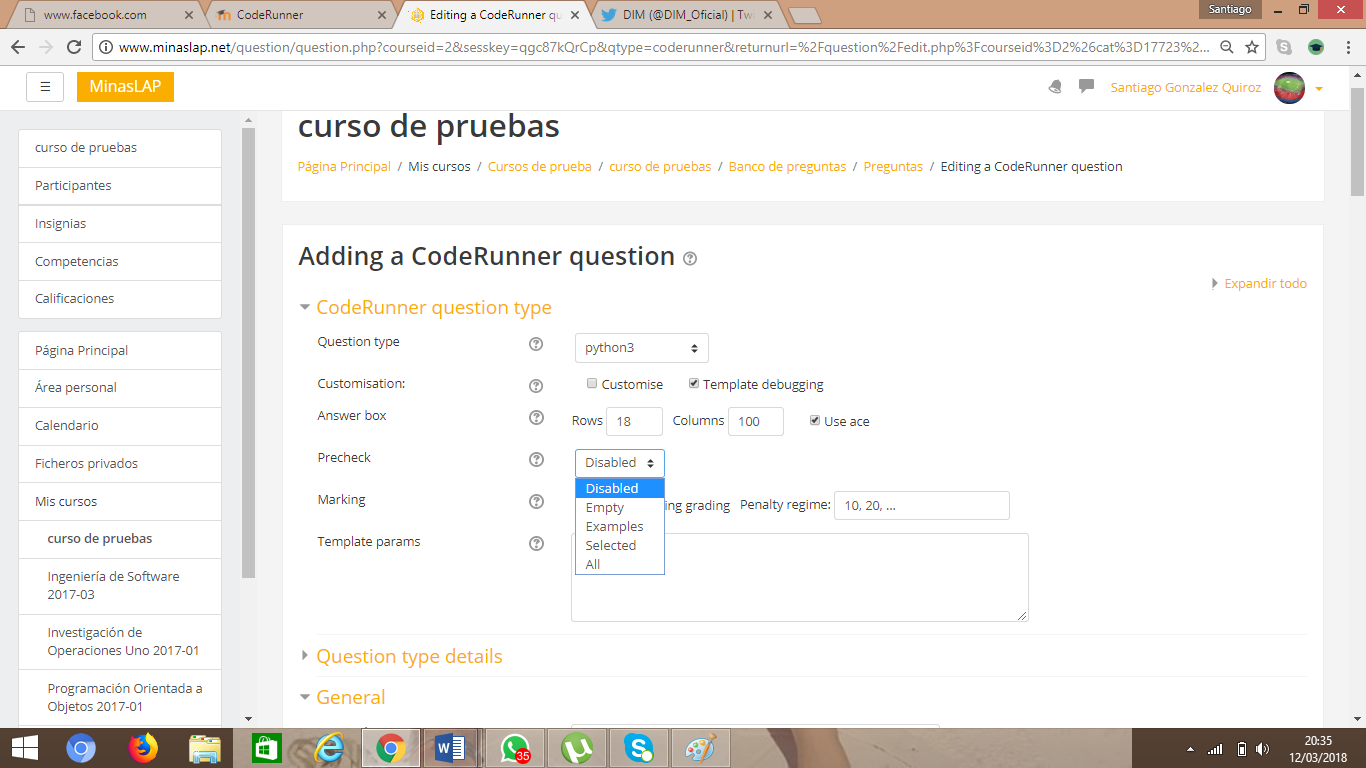
Al no chequear el customisation, no aparecerá en pantalla el desplegable customisation, entonces se obtendrá algo así:



Donde se podrá pasar directamente a definir la pregunta. Recordar que entonces por defecto vendría con el grading en respuesta exacta.

Ahora se informará sobre otros parámetros también interesantes dentro de la definición de una pregunta:

* ***Answer box*:** Este parámetro permite configurar cuantas filas y cuantas columnas tendrá la caja de respuesta donde el estudiante escribirá su código. Si la respuesta del estudiante excede el límite, aparecerá la barra para deslizarse horizontalmente por la pantalla.
* ***Precheck*:** El *precheck* es una opción que puede ser útil para los estudiantes, ya que funciona como una “Prueba” sin calificación del ejercicio. Cuando el *precheck* se selecciona, se puede escoger una de las siguientes opciones:

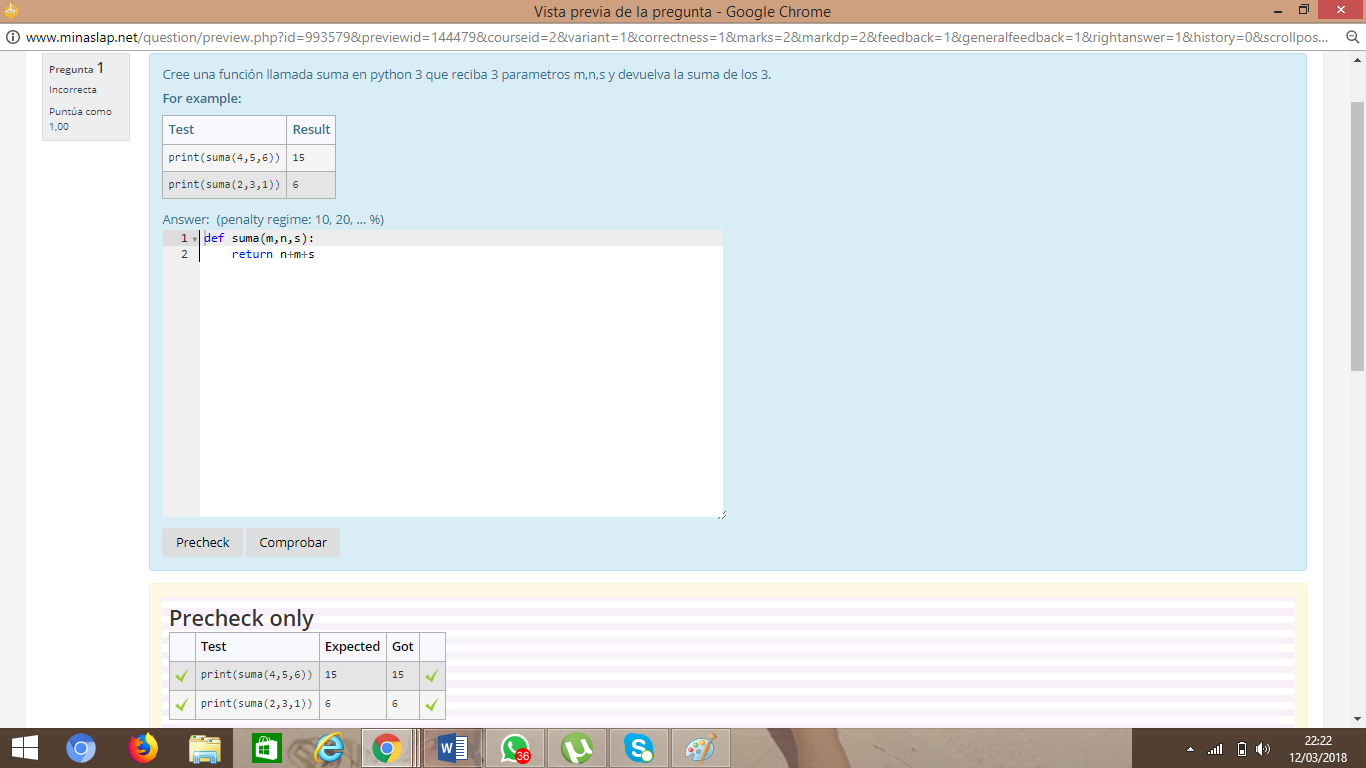


**\**Disabled*:** Quiere decir que el precheck no está activado y, por ende, el estudiante **NO** podrá hacer una prueba “gratis” de su ejercicio.

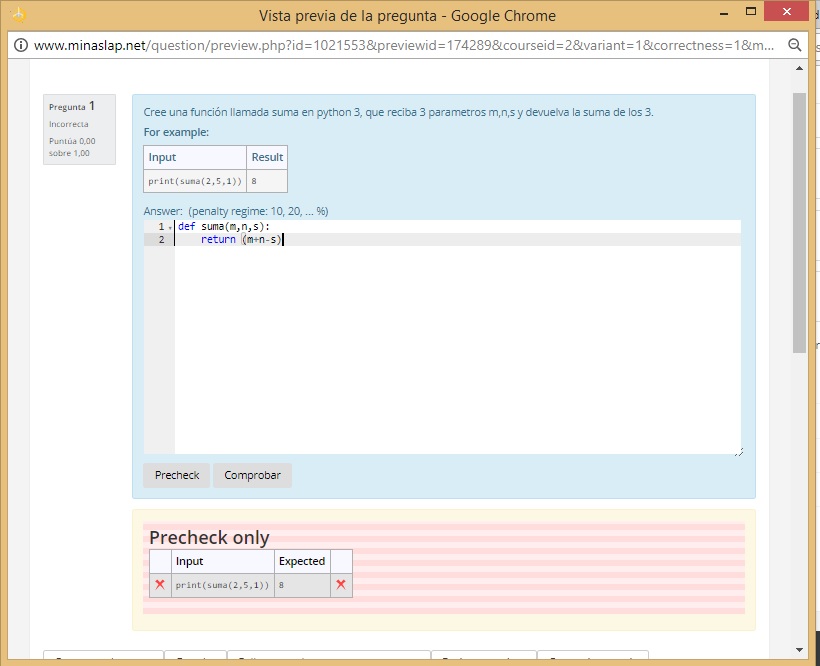
**\**Examples***: Quiere decir que el precheck se realiza con unos casos de prueba definidos como “Example” (Más adelante se explican los casos de prueba).

**\**Selected***: El precheck se realizará con algunos casos de prueba seleccionados.

**\**All*:** El precheck se realiza con todos los casos de prueba establecidos.



Esto es lo que obtiene un estudiante al realizar el precheck de se pregunta y dicho precheck, pasa los casos de prueba seleccionados.

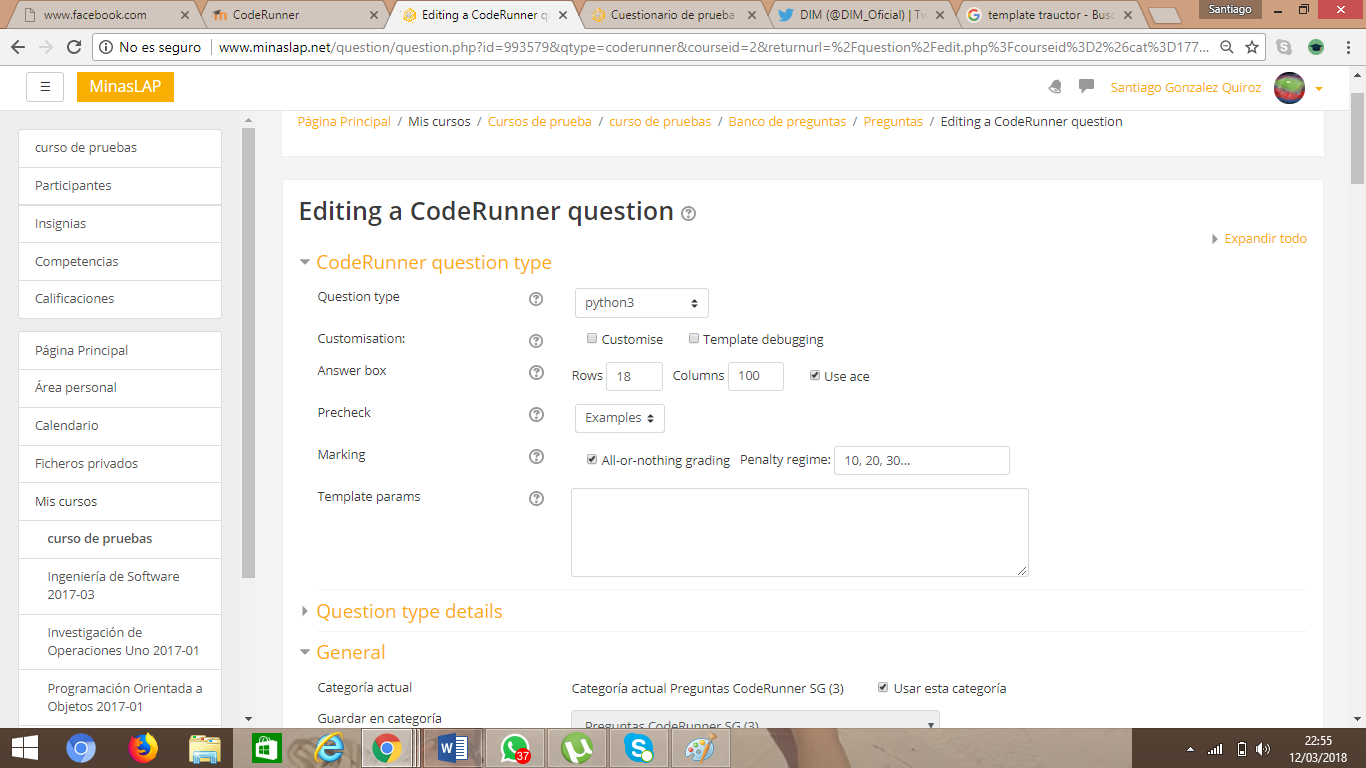


Esto es lo que obtiene el estudiante al fallar en el precheck.

**Vale aclarar que el hecho de que un precheck funcione, no le asegura al estudiante que su respuesta esté correcta, pues recordemos que el precheck se realiza con base en un caso de prueba especificado, pero podría fallar con otro caso.**

* **M*arking*:** El *marking* es la penalidad que aplicara el CodeRunner a un estudiante en

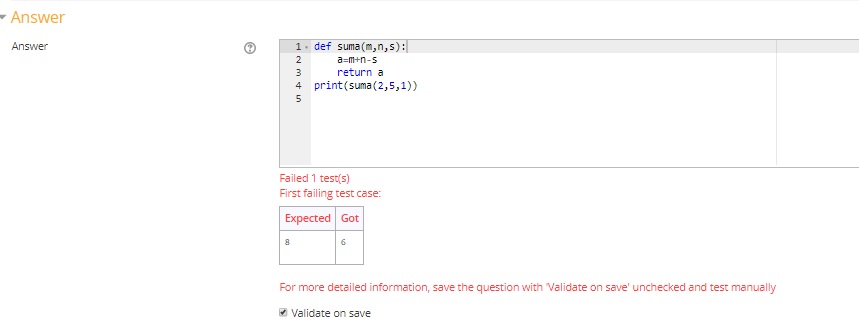
su nota por cada envió erróneo que haga del código. El *marking* se aplica progresivamente, haciendo cada vez más fuerte la penalización. Cuando el “*all or nothing*” esta seleccionado, quiere decir que todos los casos de prueba deben ser pasados con éxito para generar alguna nota. El 10, 20, 30, quiere decir que en primer envió erróneo se cobra un 10% de la nota, en el segundo un 20% y así sucesivamente. **Podría configurarse en 0%, para que el estudiante no tenga ninguna penalización en su nota por cada envió erróneo.**



* **General:** Dentro de la categoría general se definen cosas muy básicas como el nombre de la pregunta y una breve descripción, dentro de esta categoría es importante el parámetro “*Answer*”.

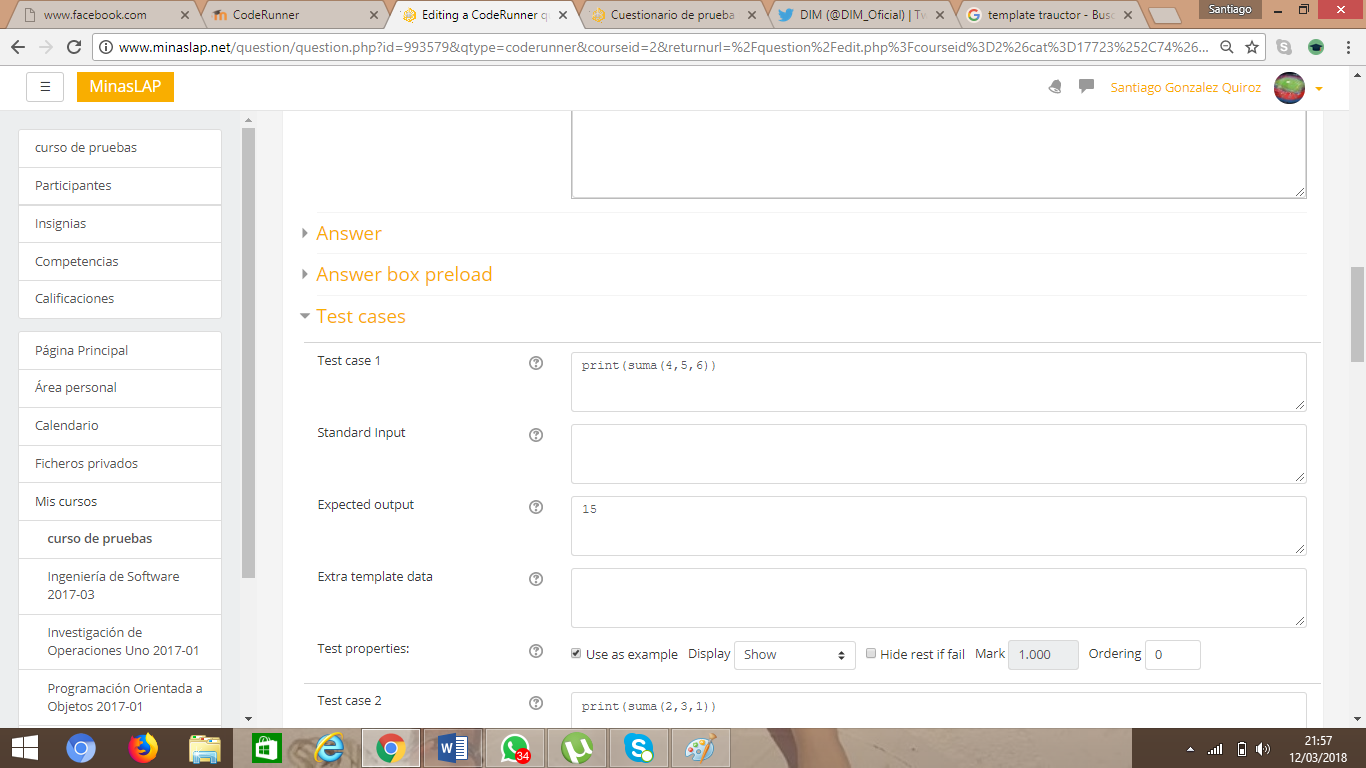
**El parámetro *Answer*** es opcional para una pregunta, pero es verdaderamente útil para evitar un error por parte de quien diseña el cuestionario. En esta caja *answer*, lo que se ingresa es lo que se desea que el estudiante responda, tal cual como él lo haría. Después de poner la respuesta deseada, se selecciona la opción “*Validate on save*”. Cuando se guarda la pregunta y la opción “*Validate on save*” esta seleccionada, CodeRunner evalúa la respuesta puesta por el profesor con el **caso de prueba seleccionado como ejemplo**, si esta respuesta no pasa la prueba, **no** permite que el profesor guarde la pregunta, ya que evidentemente tendría un error. Solo la deja guardar cuando pasa los casos de prueba.

Acá se muestra un caso donde el CodeRunner no deja guardar la pregunta, ya que la respuesta proporcionada por el profesor no pasa el caso de prueba seleccionado como ejemplo.



Efectivamente, el programa esperaba un 8 como salida y recibió un 6, por ende, no deja que el profesor guarde la pregunta.

* **Casos de prueba (MUY IMPORTANTE):** Para una pregunta se pueden crear el número que se quiera de caso de prueba, lo importante es que tipo de caso de prueba es. En la caja de test case, se ingresa el código con el que voy a poner a prueba la respuesta del estudiante. Por ejemplo, si el estudiante debía crear una función en Python que se llame suma y le retornara la suma de tres parámetros n, s, m entonces en el test case se pondría algo de la forma: “print (suma (4,5,6))”, lo cual llamaría a la función suma con los parámetros 3,4,5 y retornaría la suma de ellos para luego imprimirla. Como podemos ver en la imagen:

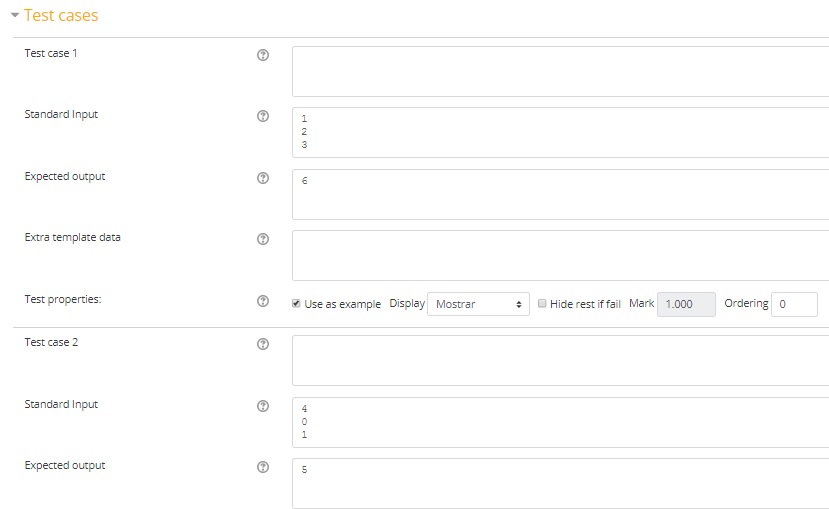


Nótese el caso de prueba.

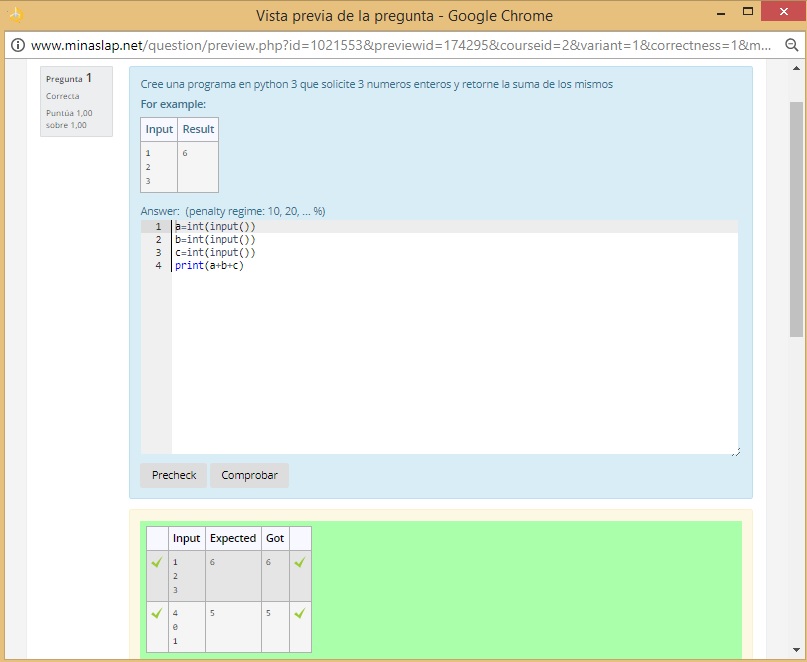
**Adicionalmente en el cajón de *expected output*** se aclara el valor que esperamos que nos imprima dicho caso, será este valor el que el CodeRunner evaluara por medio del *template* y del *grading* para asignar una nota.

***Standard input*:**

Sirve para hacer casos de prueba en los que se quieren entradas específicas, solo tenemos que añadir cada una de las entradas por línea. Se observa este ejemplo, en el cual queremos un programa que solicite 3 números y nos imprima la suma de los mismos. Se definen los siguientes casos de prueba:



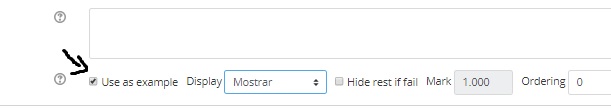
Observamos 2 casos de prueba con 3 entradas cada uno y una salida esperada. (Las entradas siempre separadas cada una por línea). Miremos que pasa en ejecución:



Efectivamente el programa hace bien las comparaciones con las entradas de los casos de pruebas y las salidas.

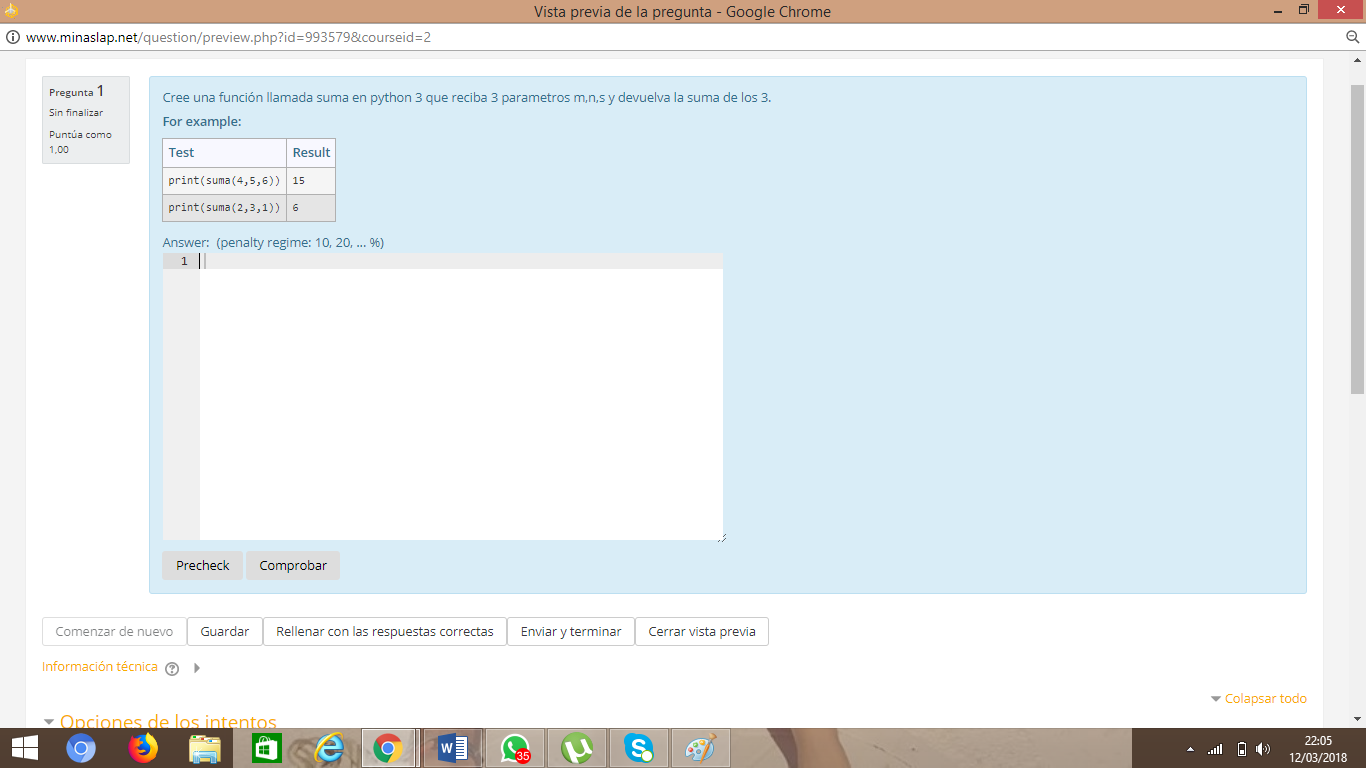
Ahora viene otro aspecto importante dentro de los casos de prueba y son las propiedades, las cuales van a definir qué tipo de caso de uso es y si es visible para el estudiante o no.

**\**Use as Example*:** Sirve para poner un caso de prueba como ejemplo, al establecerlo como ejemplo, será el que el CodeRunner va a probar al seleccionar la opción “precheck” con *examples*, también será el que el programa testeara cuando el diseñador de la pregunta selecciona el “*Validate on save*” en *answer* y además será un ejemplo visible para el estudiante dentro de la pregunta:



Basta con chequear el recuadro “*Use as example*” para establecer un caso de prueba de este tipo.

Así se visualizaría el caso de prueba seleccionado:



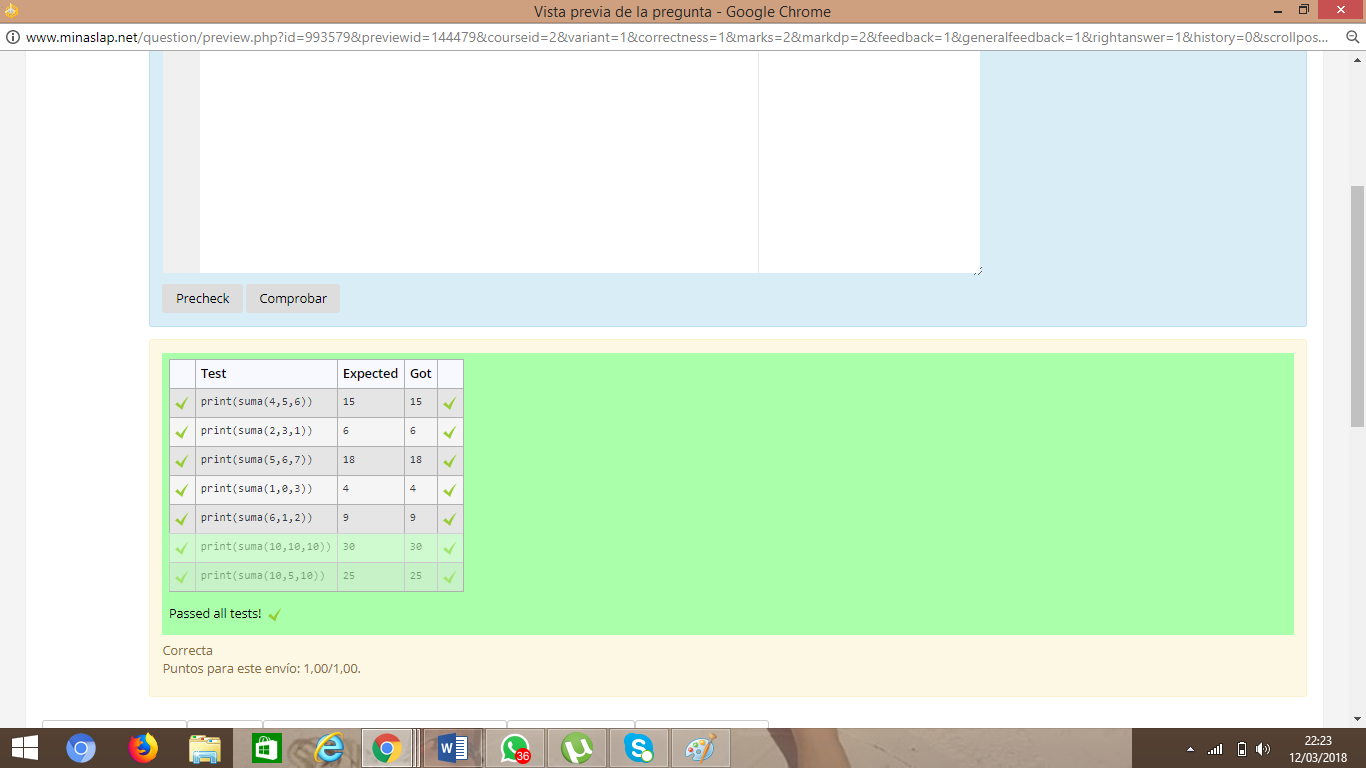
Para la pregunta de la imagen anterior se crearon dos casos de prueba del tipo “*Use as example*” que como se puede ver, a los estudiantes les aparece en la parte superior de la pregunta. Los casos de prueba que no estén definidos como ejemplo NO aparecerán en esta parte. Es posible no definir casos de prueba como ejemplo y que por ende no aparezca ninguno en la parte superior.

**\**Display*:** El *display* es un parámetro muy importante dentro de los casos de prueba, ya que es el medio por el cual se selecciona cuales pruebas que le hace el CodeRunner al código son visibles por el estudiante. Dentro del display hay 4 opciones: Show (Mostrar la prueba), *Hide* (Ocultar la prueba, solo la ven los profesores), *Hide if fail* (Ocultar si falla la prueba), *hide if succeed* (Ocultar si pasa la prueba). Cuando un caso de prueba está marcado como show, siempre será visible para los estudiantes después de que la plataforma evalúe su código, se pone en verde si pasa dicho caso de uso y en rojo cuando no lo pasa. Cuando un caso de uso está en *hide*, nunca será observado por el estudiante, solo por los maestros y también será verde o rojo según si fue aprobado o no. Se observa lo anterior por medio de una imagen:

Aquí seleccionamos el tipo de display:

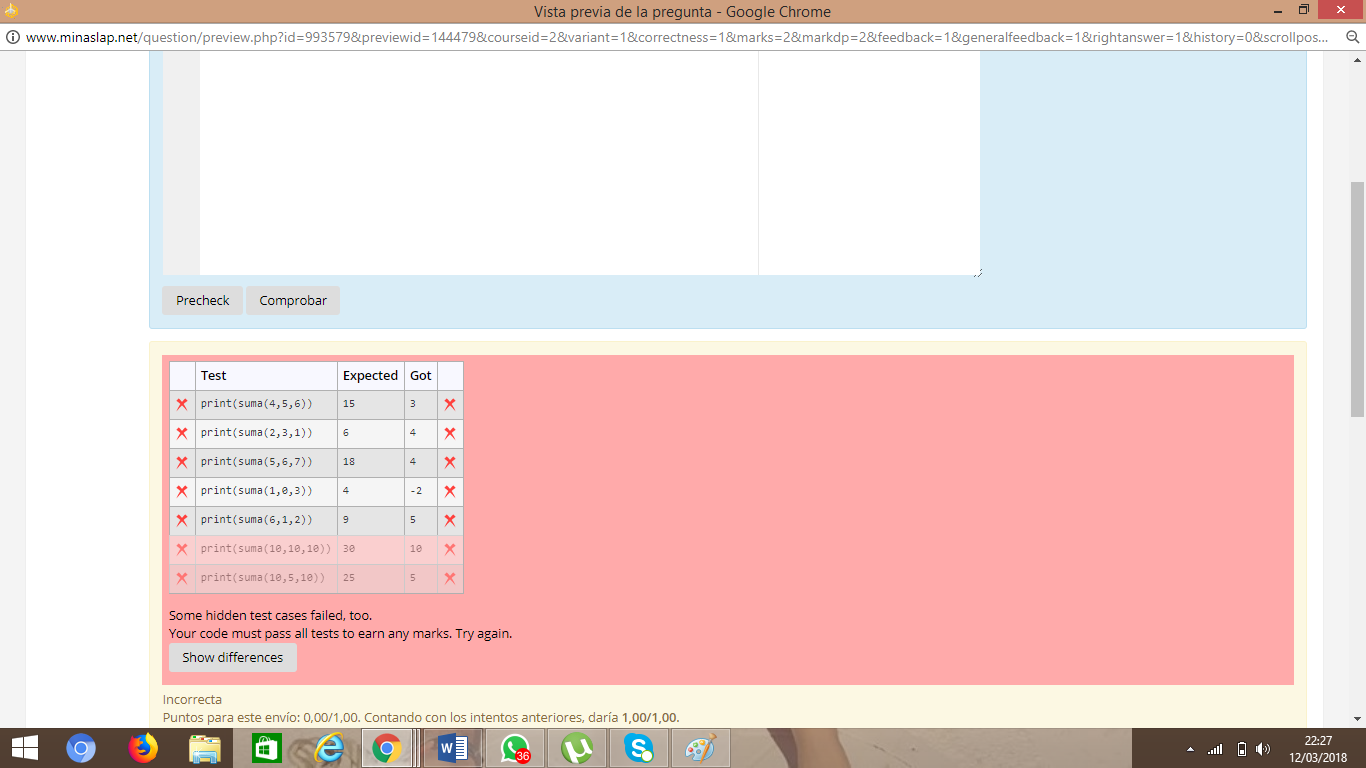


Aquí observemos lo que pasa con cada uno de los tipos:



Como puede verse, la imagen corresponde a la retroalimentación recibida por un estudiante después de enviar su respuesta, los casos de prueba sombreados con gris, quiere decir que son casos seleccionados como “*Hide*”, por lo que los profesores pueden verlo, pero el estudiante no tiene acceso a dicha información. Los casos de prueba que no están sombreados si son visibles por los estudiantes. El sombreado verde indica que casos de prueba fueron pasados por el código, en este caso todos fueron pasados.

Se observa una ocasión donde no se pasen los casos de prueba:



Igualmente, los estudiantes no ven los casos marcados como *hide*.

Así es como ve exactamente la misma pregunta un estudiante (Desde perfil de estudiante):



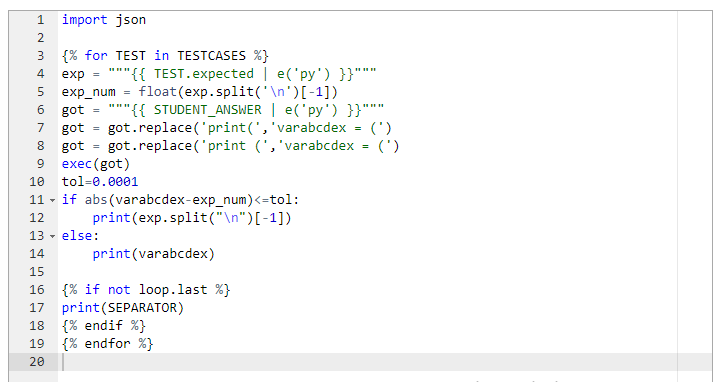
Se puede observar que ya solo se muestran 5 casos de prueba, ya que los dos que están marcados como hide, solo son visibles para los maestros.

**Perfectamente podría definirse una pregunta donde todos los casos de prueba se marquen como hide y el estudiante no pueda ver ninguno.**

**Anexos: actualizado 03/03/2019**

**Plantilla para no tener problemas con redondeos:**

Después de haber diseñado muchas preguntas, se pudo notar que en varias ocasiones había problemas con las preguntas que dentro de sus cálculos numéricos hacían redondeos. Particularmente el problema se presentaba con el redondeo que se hacía cuando un resultado daba 0 (cero) o algún número entero (2.0, 3.0..). Esto se prestaba para confusión debido a la parte decimal del entero, ya que las respuestas programadas en cada pregunta son sensibles y toman un 4.0 diferente a un 4. Para corregir este problema, se hizo una modificación a la template de evaluación, la cual corrige dicho problema. La plantilla es:



**import json**

**{% for TEST in TESTCASES %}**

**exp = """{{ TEST.expected | e('py') }}"""**

**exp\_num = float(exp.split('\n')[-1])**

**got = """{{ STUDENT\_ANSWER | e('py') }}"""**

**got = got.replace('print(','varabcdex = (')**

**got = got.replace('print (','varabcdex = (')**

**exec(got)**

**tol=0.0001**

**if abs(varabcdex-exp\_num)<=tol:**

**print(exp.split("\n")[-1])**

**else:**

**print(varabcdex)**

**{% if not loop.last %}**

**print(SEPARATOR)**

**{% endif %}**

**{% endfor %}**

**Cómo evaluar un ejercicio del tipo crear función?**

Cuando se desea evaluar un ejercicio del tipo crear una función, hay que tener en cuenta una modificación importante en el diseño de la pregunta, precisamente en el campo de los test cases, en el subcampo test case. Vale recordar que este subcampo test case, es simplemente un trozo más de lenguaje (en el que se esté programando), que se añade al final de la respuesta del estudiante. Este campo es importante, porque es aquí donde se realiza el llamado a la función que el estudiante previamente debió haber programado. Con esto se logra que el estudiante efectivamente realice una función, más no resuelva el ejercicio sin ella.

Ejemplo:

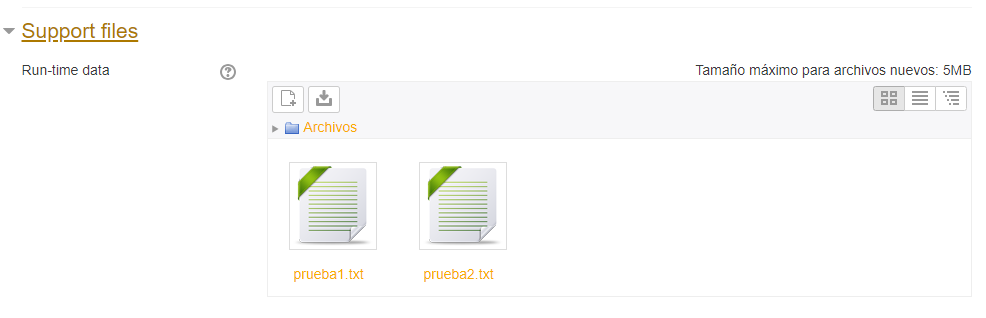


En este ejemplo al estudiante se le pidió realizar una función llamada Binario, la cual recibe un parámetro string y lo convierte a decimal. En el campo test case, se observa cómo se realiza el llamado a la función Binario(“100101”). Con esto ya se obliga al estudiante a tener que realizar la función. Si la función no es programada por el alumno, al hacer el llamado a la misma en el test, inmediatamente generaría un error porque no existiría y posteriormente la nota seria reprobada.

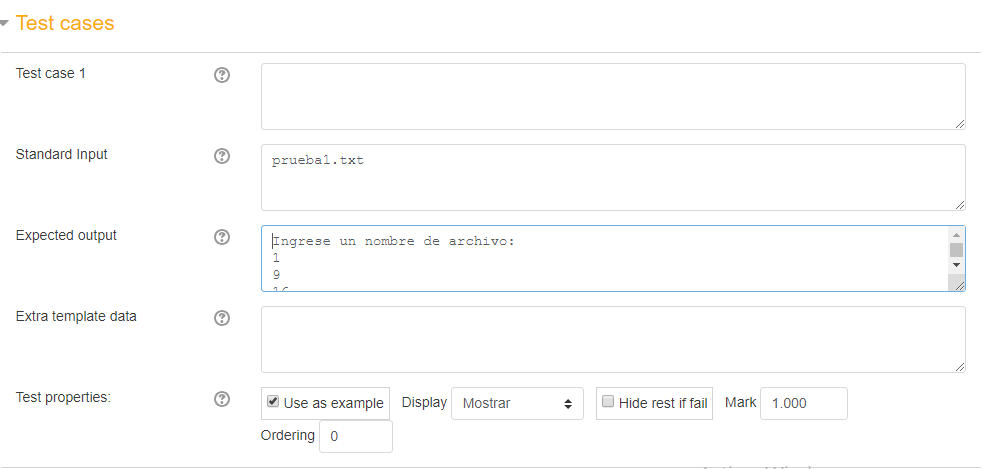
**Cómo evaluar ejercicios con escritura y lectura de archivos?**

El manejo de archivos es un aspecto importante dentro de la programación, por eso es natural que se quieran hacer evaluaciones de este tipo y acá se explica cómo realizarla. La parte fundamental en este tema es: Cuales son los archivos que voy a leer/manipular? Estos archivos deben existir dentro del servidor, deben ser cargados a la plataforma por parte de quien diseñe la pregunta. Acá se explica donde cargar estos archivos:

Los archivos se cargan en el apartado support files, que está después de los test cases. En esta sección se selecciona la opción “agregar” y en el explorador se buscan los archivos que se quieren usar. Se pueden añadir varios archivos, es conveniente agregar un archivo por cada test case de la pregunta. Después de cargados, se procede a guardar y ya es posible usarlos durante la evaluación.



Siguiendo con el ejemplo de la imagen anterior, los test case serian algo así:



En este caso, al estudiante se le pidió solicitar un String con el nombre del archivo que se desea abrir, por lo que en el input se le indica el nombre del archivo previamente cargado. Vale aclarar que el archivo con el que se va a trabajar y el archivo de respuesta del estudiante, están en la misma ubicación en el servidor, por lo que no hay que preocuparse por la ruta.

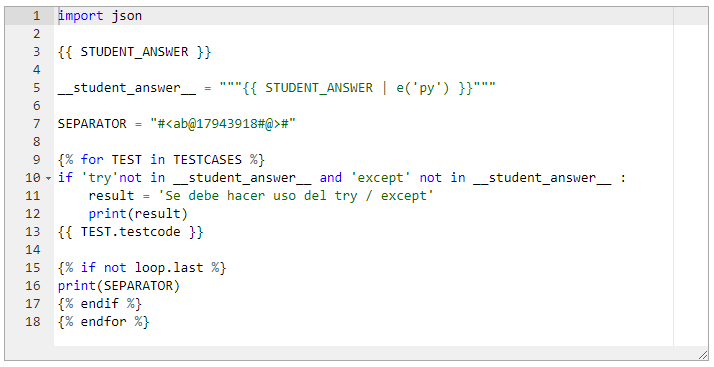
**Bonus:** Es posible combinar ejercicios de manejo de archivos con ejercicios de funciones. Puede que se desee que el estudiante lea un archivo para posteriormente usar sus datos en alguna función. Una combinación de esos dos casos seria algo como:



Se puede observar como además de hacer el llamado a una función en el test case, se proporciona un nombre de archivo en el input.

**Cómo evaluar ejercicios con el try/except (o sus análogos en los diferentes lenguajes)?**

El manejo de errores es otro factor importante dentro de la programación, por eso aquí se explicara como diseñar preguntas para evaluar esta temática.

En muchas ocasiones, un error se puede manejar por medio de condicionales. Si por ejemplo yo quiero manejar un error de dividir por cero, cuando trabajo con dos variables, bastaría con poner un condicional tipo: if variablex==0: print(“No se puede dividir por cero”). Esto no está mal, pero si un profesor lo que en realidad quiere es evaluar el uso del manejo de errores por medio de try/except, podría obligar al estudiante a usar dicha estructura, o el ejercicio le saldría con error. Esto se puede lograr manipulando la template de evaluación por la siguiente: 

**import json**

**{{ STUDENT\_ANSWER }}**

**\_\_student\_answer\_\_ = """{{ STUDENT\_ANSWER | e('py') }}"""**

**SEPARATOR = "#<ab@17943918#@>#"**

**{% for TEST in TESTCASES %}**

**if 'try'not in \_\_student\_answer\_\_ and 'except' not in \_\_student\_answer\_\_ :**

**result = 'Se debe hacer uso del try / except'**

**print(result)**

**{{ TEST.testcode }}**

**{% if not loop.last %}**

**print(SEPARATOR)**

**{% endif %}**

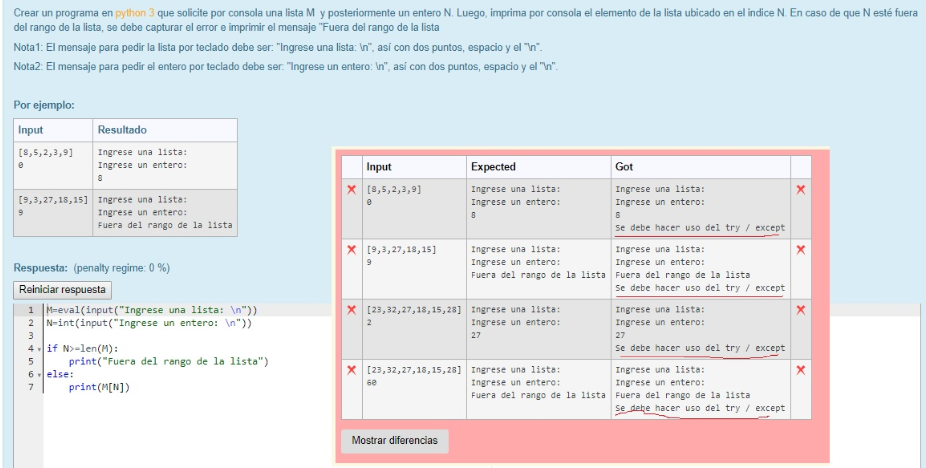
**{% endfor %}**

Esta plantilla lo que hace es que busca dentro de la respuesta del estudiante si en algún lado aparece el “try” y/o el “except”. En caso de no encontrar estos componentes en la respuesta del estudiante, automáticamente hará una impresión por pantalla de un mensaje no esperado por los output del ejercicio, lo cual dañaría la nota del estudiante. Esta plantilla mostrada, es un caso específico para python, pero podría modificarse fácilmente para cualquier lenguaje de programación, modificando las líneas:

**if 'try'not in \_\_student\_answer\_\_ and 'except' not in \_\_student\_answer\_\_ :**

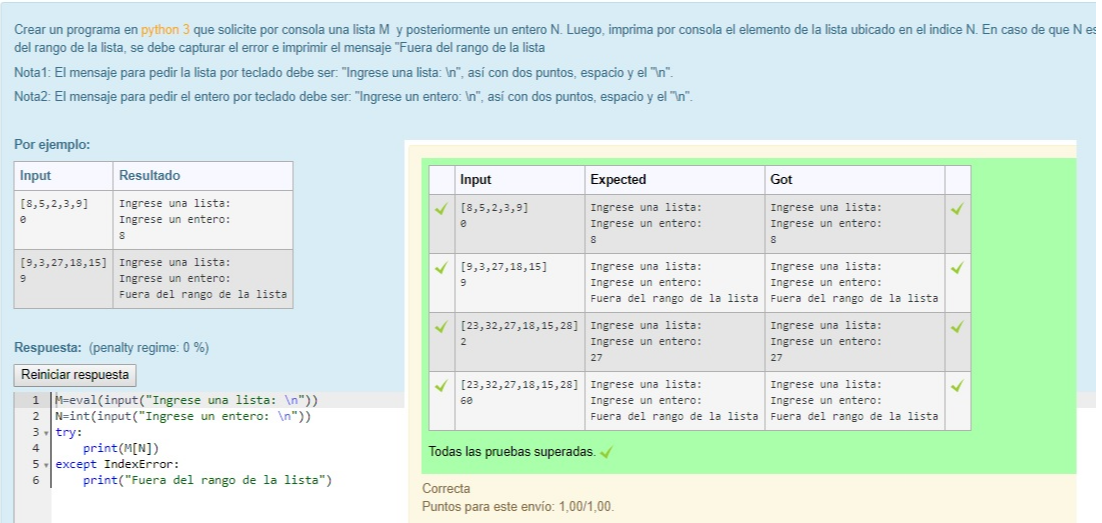
**result = 'Se debe hacer uso del try / except'**

Acá se adjuntan imágenes de lo que vería un estudiante al resolver un ejercicio sin usar try/except y lo que vería en caso de sí usarlo:



Se observa como al no usar try/except, a pesar de que el ejercicio imprime las cosas deseadas, se le califica como malo por no usar la estructura requerida.

El siguiente es un caso donde el estudiante sí lo usa:



Santiago González Quiroz. sangonzalezqui@unal.edu.co