Banco de Preguntas Parametrizadas

Luis Ernesto Carrera Retana

25 de septiembre de 2020

1. Información general

Este proyecto consta de los siguientes programas:

- `generar` Esta es la función que genera las pruebas. Requiere dos archivos como argumentos: el primero es el archivo `ppp` donde se guarda la información general de la prueba, y el segundo la carpeta con las listas de los grupos de las personas estudiantes (cada uno de ellos descargado del tec digital y convertido a CSV) o un solo archivo csv. En esta misma carpeta guarda las evaluaciones.

- `evaluar` Esta es la función que realiza la evaluación. Requiere el archivo `ppp`, la carpeta con las listas de los estudiantes, y el archivo de las respuestas descargado de Microsoft Forms, y convertido a .csv, con ';' como separador entre columnas.

- `visualizar` Esta función recibe como argumento la dirección del archivo de una pregunta, el número de ejemplos por generar, para generar un pdf de la pregunta. Esto permite revisar que la pregunta esté bien definida, y ver si genera los resultados esperados.

  Si la pregunta requiere de paquetes adicionales, de manera opcional se puede pasar como argumento un *encabezado* de la pregunta que debe iniciar desde `\documentclass...` y llegar hasta `\begin{document}`. Debe incluir un ambiente de tipo *ejer*, porque cada pregunta se encierra entre un `\begin{ejer}` ... `\end{ejer}`. Puede ver la variable `encabezado` en el archivo `visualizar.py` como referencia.

  Cuando se imprime la vista previa de preguntas de selección, el orden de los distractores se mantiene fijo según el orden original de la pregunta y se marca con 'R/' la opción correcta. En el caso de preguntas de respuesta corta, se agrega la respuesta al final de la pregunta.

1.1. Requisitos

- python
- LaTeX(miktex o texlive)
- perl (no se requiere para la función `visualizar`)
- ghostscript (no se requiere para la función `visualizar`)

1.2. Modificando el LaTeX

El encabezado predeterminado se puede visualizar en el archivo `latex.py`. Recuerde respaldar cualquier cambio que haga, porque cada versión nueva va a borrar los cambios que usted haya realizado.

Se utiliza el español como paquete predeterminado de idiomas. Si quiere utilizar punto decimal en lugar de la coma, puede incluir en

```
<Encabezado>
\decimalpoint
```

2. Estructura del archivo general

```
<Escuelas>
__Lista de escuelas__

<Semestre>
__Semestre y año__

<Tiempo>
__Duración de la prueba__

<Cursos>
__Nombre de los cursos__

<Titulo>
__Título de la prueba__

<Encabezado>
__Paquetes, comandos nuevos, etc, para el encabezado del archivo LaTeX__

<Instrucciones>
__Instrucciones del examen__

<Seccion[, orden = aleatorio]>
  <Titulo>
  __Opcional si es sólo una sección__

  <Instrucciones>
  __Opcional si es sólo una sección__

  <Preguntas>
    <bloque>
    [puntaje = __int__,] origen = __string__[, muestra = __int__]
    </bloque>
<Fin>
```

- Todo texto entre `[ ... ]`, a excepción de las listas de python, se refiere a argumentos opcionales.

- Tanto el nombre del curso como el título deben ser solamente **una** línea de texto debajo de la etiqueta respectiva.
- Las instrucciones pueden abarcar varios renglones e incluir líneas en blanco, para que LaTeX separe los párrafos.
- En las instrucciones *no* debe aparecer el símbolo de abrir etiquetas (ver `abrir` en `Info.py`) como primer carácter del párrafo, porque es la forma de determinar que allí finalizan.
- No debe haber espacios en blanco entre las especificaciones de las preguntas, pero sí se permiten comentarios.
- El `origen` de la pregunta puede ser un archivo con extensión `.tex` o una carpeta.
- Si la dirección es una carpeta, entonces el puntaje para cualquiera de las preguntas es el mismo. Si el puntaje no aparece, entonces el predeterminado es 1 punto.
- La muestra se refiere al número de preguntas que se toma de la carpeta. Si no aparece, el predeterminado es 1.
- Las preguntas finalizan con una línea en blanco.
- De manera opcional, se pueden encerrar varias preguntas entre `<bloque>` y `</bloque>`. El efecto es que no va a separar las preguntas con una nueva página, y va a mantener las variables entre las preguntas. Utilizar aleatorio en la sección no afecta el orden de las preguntas en el bloque.

3. Preguntas

3.1. Encabezado

No es una pregunta. Es más bien como un texto para un grupo de preguntas. Lo ideal es que vaya al inicio de un bloque. Se distingue en el archivo `.ppp` porque tiene un puntaje de 0.

```
<tipo = encabezado[, cifras = __cifras significativas__]>

<variables>
__nombre_de_variables__ = __expresion__

<pregunta>
__texto de la pregunta__
```

3.2. Selección única

```
<tipo = seleccion unica[, opcion = (__indice__ | todos)]>

<variables>
__nombre_de_variables__ = __expresion__

<pregunta>
```

```
__texto de la pregunta__

<item>
__respuesta o distractor__
```

1. Es opcional especificar el índice de la respuesta (0-indexado); sino se asume que la respuesta correcta es el primero de los items (`__indice__ == 0`).
2. Para el caso en que haya habido un error, se puede poner `opcion = todos`, es decir, que toda respuesta es correcta.
3. **Se debe dejar una línea en blanco al final de cada item**.
4. Las variable son opcionales.

3.3. Respuesta corta

```
<tipo = respuesta corta[, opcion = todos]>

<variables>
__nombre_de_variables__ = __expresion__

<pregunta>
__texto de la pregunta__

<item[, error = __error__][, factor = __porcentaje de la pregunta__]>
__respuesta__
```

1. Para el caso en que haya habido un error, se puede poner `opcion = todos`, es decir, que toda respuesta es correcta.
2. La respuesta **no** se escribe como una @-expresión.
3. Aunque se asume que el primer item contiene la respuesta correcta, se puede especificar más de una respuesta.
4. Si no se da el error, se toma que es 0. Aunque sería lo esperable en una pregunta cuya respuesta es un entero, no se recomienda en absoluto para una respuesta de tipo flotante.
5. Si no se da un factor, entonces se toma 1 como factor. Se utiliza para ir relajando el error, pero castigar entonces el puntaje de la pregunta. Cada radio del error debe ser mayor, y cada puntaje no debe ser mayor al anterior, porque se toma como correcto el primero que se encuentre.

4. Funciones

Se tienen dos tipos de funciones. Las funciones que únicamente se pueden llamar para definir variables, y las funciones generales que se pueden llamar en las variables, en la pregunta, y en los ítemes.

En las variables se define de manera normal. En la pregunta y los items cualquier expresión que requiera ser evaluada debe escribirse como una @-expresión:

- `@<__expr__>`
- `@{__expr__}`
- `@(__expr__)`
- `@[__expr__]`
- Una `@` seguida de cualquier símbolo, que es el mismo que se utiliza para cerrar, por ejemplo podría ser `@|__expr__|`.

¡El único requisito, es que el símbolo para cerrar **no** debe aparecer en la expresión a evaluar! Bueno, no es el único. Toda expresión a evaluar debe estar contenida en una sola línea.

Si el resultado de una @-expresión es un `string` o un entero, entonces se concatena al texto; si es un punto flotante, entonces se trabaja de manera predeterminada con 3 cifras significativas, y se imprime el número en notación decimal o en notación científica, dependiendo de cómo se permita saber de la forma más clara que se tienen 3 cifras significativas. Si el resultado de la @-expresión es otra cosa, entonces se deja que python lo convierta a texto, y se concatena.

4.1. Funciones para definir variables

Estas funciones únicamente están disponibles para definir variables:

4.1. `randrange(stop)`
`randrange(start, stop[, step])`

Un elemento `n` seleccionado al azar tal que `start<=n<stop`.

4.2. `randint(a,b)`

Un elemento `n` seleccionado al azar tal que `a<=n<=b`.

4.3. `choice(__seq__)`

Un elemento `n` seleccionado al azar de la sucesión no vacía `seq`.

4.4. `sample(<list>, k)`

Toma una muestra de tamaño `k` de la lista `list`. `k` debe ser menor o igual al tamaño de la lista. La muestra no está ordenada con respecto a la lista. Si `k` es el mismo tamaño de la lista, entonces devuelve una permutación de la lista.

4.5. `random()`

Genera un número aleatorio con distribución uniforme en el intervalo $[0, 1)$.

4.6. `uniform(a, b)`

Un elemento `x` que sigue una distribución uniforme tal que `a<=x<=b`.

4.7. `gauss(mu, sigma)`

Un elemento `x` que sigue una distribución gaussiana con media `mu` y desviación estándar `sigma`.

4.2. ¿Cómo construir una sucesión?

En python las sucesiones para la función `choice` o `sample` se pueden definir mediante una de las siguientes formas:

- `range(<stop>)`
- `range(<start>, <stop>[, <step>])`

1. Si la sucesión está dada por los j elementos $0, 1, 2, \ldots, j-1$, se construye con `xs = range(j)`.
2. Si la sucesión está dada por los j elementos: $i, i+1, i+2, \ldots, i+j-1$, se construye con `xs = range(i, i+j)`.
3. Para una sucesión aritmética de k elementos $i, i+d, i+2d, \ldots, i+(k-1)d$, se construye con `xs = range(i, j, d)`, donde $i+(k-1)d < j \leq i+kd$.
4. Para concatenar dos o más sucesiones:

   `xs = [*range(<start>,<stop>[,<step>]), *range(<start>,<stop>[,<step>])]`.

Si lo que se quiere es tomar un elemento aleatorio de una sucesión simple, entonces mejor utilizar las funciones `randrange` o `randint`.

4.3. Constantes y funciones generales

Lo que se tiene es un subconjunto de constantes y funciones de python, y algunas programadas específicamente para la generación de pruebas.

4.8. `div`; `inf`; `pi`

El valor de π, y constantes utilizadas para evaluar límites en respuesta corta. Así $-\infty$ se escribiría `-inf` y $+\infty$ como `inf` o `+inf`. Si el límite sea divergente entonces utilizar `div` (internamente se trabaja con la constante `math.nan`, *not a number*).

La constante `e` se incluye como parte de la biblioteca de matemática: `math.e`.

4.9. `lambda x: __funcion que depende de x__`

Funciones anónimas. Para que funcione, todas las variables deben definirse como argumento. Por ejemplo, para definir una función cuadrática con parámetros a, b, c generados de manera aleatoria, se podría hacer así:

```
<tipo = respuesta corta>

<variables>
a, b, c = sample([*range(-9, 0), *range(1, 10)], 3)
cuad = lambda x, a, b, c: a * x**2 + b * x + c

x = randint(-3, 3)

<pregunta>
Considere $f(x) = @|txt.coef(a)|x^2 @|txt.coef(b, True)|x @|txt.conSigno(c)|$.
```

```
Determine $f(@|x|)$.

<item>
cuad(x, a, b, c)
```

Esta implementación no permite el uso de funciones recursivas. Sin embargo se puede utilizar un combinador-Y para lograrlo. Ello consiste en agregar una función como parámetro, y luego, a la hora de llamar a la función, pasarse a sí misma como argumento. Veamos un ejemplo:

```
<tipo = respuesta corta>

<variables>
i = randrange(3)
h = [ math.cos, math.sin, lambda x: math.exp(-x)][i]
txt = ['\cos(x)', '\sin(x)', 'e^{-x}'][i]
x0 = [0, 1, 2][i]
n = randint(2, 5)

pf = lambda g, f, x0, n: x0 if n == 0 else g(g, f, f(x0), n - 1)

<pregunta>
Determine el valor de realizar @|n|@ iteraciones del punto fijo de la función
$f(x)=@|txt|$ iniciando en @|x0|.

<item, error = 0.01>
pf(pf, h, x0, n)
```

4.10. `len(xs)`, `sorted(xs)`

Devuelven respectivamente la longitud de un iterable, y el iterable de manera ordenada creciente. Si se quiere ordenar `xs` de manera decreciente, entonces se debe escribir `sorted(xs, reverse = True)`. También se puede utilizar una función para especificar la función de orden:
`sorted(xs, key = __funcion__)`. Por ejemplo:

```
ls = [*range(-9, 10)]
ls = sorted(sample(ls, 5), key = abs)
```

podría dar como resultado `[2, 4, -5, -6, 8]` para `ls`.

4.11. Funciones de matemática:

a) `+ - * / // % **`

Suma, resta, multiplicación, división normal (con punto flotante como respuesta), división entera, residuo y potencia.

b) `degrees(x)`; `radians(x)`

Conversión de radianes a grados; conversión de grados a radianes, π.

c) `math.acos(x)`; `math.acosh(x)`; `math.asin(x)`; `math.asinh(x)`; `math.atan(x)`; `math.atan2(y, x)`; `math.atanh(x)`; `math.cos(x)`; `math.cosh(x)`; `math.sin(x)`; `math.sinh(x)`; `math.tan(x)`; `math.tanh(x)`

Funciones trigonométricas y sus inversas (en radianes).

d) `math.isqrt(x)`; `math.sqrt(x)`; `math.exp(x)`; `math.log(x)`; `math.log10(x)`

Parte entera de la raíz de `x`; funciones en punto flotante para la raíz, exponencial, logaritmo natural y logaritmo en base 10.

e) `erf(x)`; `erfc(x)`; `gamma(x)`

Funciones de error y función gamma.

f) `fmod(x, y)`; `modf(x)`

Función residuo para variables de tipo flotante; parte decimal y parte entera de x.

g) `round(__numero__[, __dig__])`

Redondea un número, y se puede especificar el número de dígitos a utilizar. 0 es el valor predeterminado.

h) `math.ceil(x)`; `math.floor(x)`; `math.trunc(x)`

Funciones de manejo de decimales.

i) `math.factorial(a)`; `math.comb(n, r)`; `math.perm(n, r)`

Factorial, combinaciones y permutaciones.

j) `math.hypot(x, y)`; `math.dist(xs, ys)`

$\sqrt{x^2+y^2}$ y distancia entre los vectores n-dimensionales `xs` y `ys`

k) `sum(xs)`; `math.prod(xs)`

Suma y producto de los elementos de la secuencia `xs`.

4.12. `__exp_True__ if __exp_bool__ else __exp_False__`

Un `if` en un sólo renglón. En las expresiones booleanas se puede utilizar `not`, `or` y `and` según se necesite.

4.13. `pow(a, b), a**b` a^b

4.14. `math.gcd(a,b)` Máximo común divisor de a y b.

4.15. `mate.factores(a)`

Factorización de a. Por ejemplo `factores(1000)` devuelve la lista `[(2, 3), (5, 3)]`, que representa a $2^3 \cdot 5^3$.

4.16. `txt.frac(a, b[, conSigno=False])`

Simplifica y escribe la fracción respectiva en LaTeX usando `dfrac`. `conSigno` es un booleano (`False` o `True` con `False` como predeterminado) que escribe un signo + antes de la fracción si es positiva.

4.17. `txt.raiz(a[, n=2[, conSigno=False]])`

Simplifica y escribe la raíz respectiva en LaTeX. `conSigno` es un booleano (`False` o `True` con `False` como predeterminado) que escribe un signo + antes de la raíz si es positiva.

4.18. `txt.coef(a[, conSigno=False])`

Coeficiente que precede a una variable entera. Si $a = 1$ entonces no se escribe (o se escribe solo un signo + si `conSigno`). De manera similar ocurre si $a = -1$. En caso contrario imprime el valor.

4.19. `txt.conSigno(n)`

Imprime el número entero con signo.

4.20. `txt.expo(n)`

Si `n == 1` no imprime nada. Si no imprime `^{n}`.

4.21. `txt.numero(n)`

Para $n > 0$, devuelve el texto que representa al valor de n.

4.22. `txt.decimal(n, cifras, conSigno = False)`

Imprimir un número según las cifras dadas.

4.4. Trabajando con sucesiones

En python las sucesiones son 0 indexadas. Así, si `xs` es una sucesión, entonces `xs[0]` es el primer elemento. El tamaño de una sucesión se da por `len(xs)`. El último elemento sería `xs[-1]`. Un subconjunto: `xs[i:j]` va a ser los elementos que hay desde `xs[i]` hasta `xs[j-1]`.

4.5. Listas por comprensión

```
xs = [i for i in range(10)]
ys = [random() for i in range(10)]
ls = sorted([(xs[i], ys[i]) for i in range(10)], key = lambda tt: tt[1])
```

4.6. Formato de texto

- Si `num` es un entero, se puede imprimir el entero con signo como: `'%+d' % num`
- Si `x` es un número con decimales:
  - `'%f' % x` imprime el número con 6 decimales de manera predeterminada.
  - `'%.nf' % x`, donde `n` es un entero positivo, imprime el número con `n` decimales.
  - `'%e' % x` imprime el número en notación científica con 6 decimales de forma predeterminada.
  - `'%.ne' % x`, donde `n` es un entero positivo, imprime el número en notación científica con `n` decimales.
- Si se necesita concatenar varios textos: `'(%d, %d)' % (m, n)` representa un par ordenado de enteros.