

INF-253 Lenguajes de Programación

Tarea 1: Python

19 de agosto de 2022

1. Pixelart

Un jugador de Minecraft amante del pixelart, un día buscando creaciones de otros jugadores, se encontró con un sitio web que aseguraba tener diseños que nunca antes había visto, con instrucciones muy detalladas de como construirlos; sin embargo, este sitio no contaba con imágenes para saber como se ven los resultados de seguir esas instrucciones!

Code 1: Ejemplo de instrucciones

```
1 Ancho 8
2 Color de fondo RGB(13,181,13)
3
4 Avanzar Derecha Avanzar 2
5 Pintar Negro Avanzar
6 Repetir 2 veces { Pintar Negro Izquierda Avanzar }
7 Pintar Negro
8 Derecha Avanzar 3
9 Pintar Negro Avanzar
10 Repetir 2 veces { Pintar Negro Derecha Avanzar }
11 Pintar Negro
12 Izquierda Avanzar
13 Repetir 3 veces { Avanzar Pintar Negro }
14 Derecha Avanzar 3 Derecha
15 Repetir 3 veces { Pintar Negro Avanzar }
16 Derecha Avanzar
17 Repetir 3 veces {
18     Pintar Negro Avanzar
19     Pintar Negro Derecha Avanzar
20     Derecha Avanzar Derecha Derecha
21 }
```

Es por lo anterior que este jugador le pidió a los alumnos de Lenguajes de Programación que utilicen los contenidos del curso para crear un programa capaz de interpretar las instrucciones del sitio y producir una imagen del resultado.

Para esta tarea se debe utilizar Python 3 y los siguientes paquetes:

- [RegEx](#) para las expresiones regulares de la tarea. **Tareas que no utilicen RegEx no serán revisadas.**
- [NumPy](#) y [Pillow](#) para convertir los resultados a imágenes. El código para este proceso se les será entregado.

2. Lenguaje PixelArt (LPA)

2.1. Acerca de LPA

El Lenguaje PixelArt es un nuevo lenguaje interactivo que permite a un jugador, mediante una secuencia de instrucciones, producir uno de los pixelarts del sitio. Siempre se asume que el jugador comienza en la esquina superior izquierda apuntando hacia la derecha. El objetivo es simular el procedimiento que seguirá un jugador y guardar el resultado como una matriz (lista de listas en Python) de valores RGB (tupla de 3 valores entre 0 y 255 en Python).

2.2. Matriz

La matriz será de tamaño $n \times n$, siendo n un valor que se especificará más adelante, donde inicialmente todos los valores parten en un mismo color.

2.3. Comandos

Todo archivo **siempre** comienza con los siguientes comandos en este orden:

- **Ancho N** : Indica que la imagen será de tamaño $N \times N$.
- **Color de fondo $Color$** : Indica el color de fondo que tendrá la imagen, es decir, si un bloque no es pintado entonces será de este color. El formato que sigue $Color$ se indica más adelante.

Posterior a estos habrá una línea en blanco y luego seguirán las instrucciones, las cuales pueden ser cualquier combinación de las siguientes:

- **Izquierda**: El jugador debe girar en 90° a la izquierda.
- **Derecha**: El jugador debe girar en 90° a la derecha.
- **Avanzar N** : El jugador debe avanzar N cuadros en la dirección en la que esta mirando. En caso de que no se entregue un numero, el jugador deberá avanzar 1 bloque.
- **Pintar $Color$** : El jugador debe pintar el bloque en el que se encuentra de color $Color$. El formato que sigue $Color$ se indica más adelante.
- **Repetir N veces { $Instrucciones$ }**: El jugador debe repetir las instrucciones indicadas en $Instrucciones$ N veces. Las instrucciones pueden ser cualquiera de las anteriormente nombradas a excepción de **Ancho** y **Color de fondo**. Esto significa que pueden haber instrucciones de **Repetir** anidadas.

Cuando un comando requiera de un $Color$ como parámetro, este podrá recibir uno de los siguientes valores:

- **Rojo**: Deberá pintar de color rojo, escribiendo en la matriz de memoria la tupla $(255, 0, 0)$.
- **Verde**: Deberá pintar de color verde, escribiendo en la matriz de memoria la tupla $(0, 255, 0)$.
- **Azul**: Deberá pintar de color azul, escribiendo en la matriz de memoria la tupla $(0, 0, 255)$.
- **Negro**: Deberá pintar de color negro, escribiendo en la matriz de memoria la tupla $(0, 0, 0)$.
- **Blanco**: Deberá pintar de color blanco, escribiendo en la matriz de memoria la tupla $(255, 255, 255)$.

- **RGB(R, G, B)**: Deberá pintar del color indicado, escribiendo en la matriz de memoria la tupla (R, G, B).

A continuación se presenta un EBNF que describe formalmente la sintaxis que siguen las instrucciones de un programa:

Code 2: EBNF

```

1  instruccion ::= 'Izquierda' | 'Derecha'
2                | 'Avanzar' [numero] | 'Pintar' color
3                | 'Repetir' numero 'veces' '{' {instruccion} '}'
4
5  numero ::= 0 | no_zero {'0' | no_zero}
6
7  no_zero ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
8
9  color ::= 'Rojo' | 'Verde' | 'Azul' | 'Negro' | 'Blanco'
10         | 'RGB(' numero ',' numero ',' numero ')'

```

Pueden haber múltiples instrucciones en una línea. Adicionalmente las instrucciones de tipo Repetir pueden abarcar múltiples líneas.

2.4. Orden de operaciones

Las instrucciones siempre se ejecutan de izquierda a derecha.

3. Objetivo de la tarea

Cada estudiante debe crear un programa el cual permita recibir una cantidad indefinida de líneas en un archivo `codigo.txt`, en donde la primera línea siempre indicara el tamaño de la matriz que se utilizará, la segunda el color de fondo, la tercera será una línea en blanco y luego cada línea siguiente corresponderá a una serie de instrucciones.

Su programa debe ser capaz interpretar el código, detectar errores de sintaxis, y detectar errores de ejecución. Sintaxis: a partir del archivo `codigo.txt`, su programa debe generar un archivo `errores.txt`, donde se encontrarán todas las líneas que tengan una sintaxis incorrecta, indicando el numero de línea en la cual se encuentra el error, en el caso que no hayan errores el archivo deberá contener la frase “No hay errores!”. Intérprete: debe producir `pixelart.png` y mostrar por consola los valores de la matriz RGB en caso de que la ejecución y la revisión de sintaxis sea exitosa. Si durante la ejecución las instrucciones causaran que el jugador salga del área que cubre la matriz, se debe indicar por consola la línea que produjo este error y terminar la ejecución.

Para producir las imágenes pueden utilizar el código del siguiente GitHub Gist:
<https://gist.github.com/HectorXH/13f2c125ffbbfde381aede4a8ddbadeb>

3.1. Ejemplos

3.1.1. Ejemplo 1

Code 3: Ejemplo 1

```
1 Ancho 10
2 Color de fondo Blanco
3
4 Avanzar Derecha Avanzar
5 Repetir 4 veces {
6     Repetir 8 veces { Pintar Negro Avanzar }
7     Derecha Derecha Avanzar Derecha
8 }
```

Code 4: errores.txt

```
1 No hay errores!
```

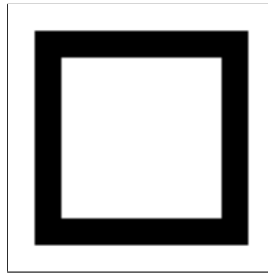


Figura 1: pixelart.png

3.1.2. Ejemplo 2

Code 5: Ejemplo 2

```
1 Ancho 3
2 Color de fondo RGB(0,0,0)
3
4 Pintar Rojo Avanzar 2 Derecha
5 Pintar Verde Avanzar 2 Derecha
6 Pintar Azul Avanzar 2 Derecha
7 Pintar Blanco Avanzar Derecha Avanzar
8 Pintar RGB(0,255,255)
```

Code 6: errores.txt

```
1 No hay errores!
```



Figura 2: pixelart.png

3.1.3. Ejemplo 3

Code 7: Ejemplo 3

```
1 Ancho 3
2 Color de fondo Negro
3
4 Repetir 2 {
5     Pintar RGB(255,0,0) Avanzar Derecha 1 Avanzar
6 }
7 Pintar RGB(255,0,0)
```

Code 8: errores.txt

```
1 4 Repetir 2 {
2 5 Pintar RGB(255,0,0) Avanzar Derecha 1 Avanzar
```

4. Sobre Entrega

- Se deberá entregar un programa llamado `pixelart.py`.
- Los ayudantes correctores pueden realizar descuentos en caso de que el código se encuentre muy desordenado.
- Las funciones implementadas deben ser comentadas de la siguiente forma. **SE HARÁN DESCUENTOS POR FUNCIÓN NO COMENTADA**

```
1  '''
2  Descripcion de la funcion
3
4      Parametros:
5          a (int): Descripcion del parametro a
6          b (int): Descripcion del parametro b
7
8      Retorno:
9          c (str): Descripcion del parametro c
10 '''
```

- Se debe trabajar de forma individual obligatoriamente.
- La entrega debe realizarse en `.tar.gz` y debe llevar el nombre: `Tarea1LP_RolAlumno.tar.gz`
- El archivo `README.txt` debe contener nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa. De no incluir `README` se realizara un descuento.
- La entrega será vía aula y el plazo máximo de entrega es hasta el 7 de septiembre.
- Por cada día de atraso se descontarán 24 pts (-1 pt por cada hora de atraso).
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Solo se contestaran dudas realizadas en AULA y que se realicen al menos 48 horas antes de la fecha de entrega original.

5. Calificación

5.1. Entrega

- Uso correcto de expresiones regulares (20 pts)
 1. No usa expresiones regulares. Utiliza otros métodos que no son los pedidos para la tarea, por ejemplo `split` e `.` (0 pts)
 2. Usa una gran y única expresión regular. Provocando así una falta de modularización de las expresiones. (MAX 12 pts)
 3. Crea diferentes expresiones y las utiliza de manera correcta, aprovechandose de la modularización generada. (MAX 20 pts)
- Detección de errores (20 pts)
 1. No detecta correctamente ningún tipo de error (0 pts)

2. Permite tan solo detectar errores simples, ya sea en la ejecución como en la revisión de sintaxis. (**MAX 5 pts**)
 3. Permite detectar errores de mayor dificultad, pero falla en instrucciones recursivas o no detecta errores durante la ejecución (**MAX 15 pts**)
 4. Detecta todo tipo de error probado, siendo no solo errores simples de sintaxis, si no que también errores de anidación de comandos o secuencias incorrectas (**MAX 20 pts**)
- Ejecución de comandos (**60 pts**)
 1. Ancho + Color de fondo **5 pts**.
 2. Izquierda + Derecha **10 pts**.
 3. Avanzar **10 pts**.
 4. Pintar color **15 pts**.
 5. Repetir veces **20 pts**.

Se asignara puntaje parcial por funcionamiento parcialmente correcto.

5.2. Descuentos

- Falta de comentarios (-10 pts c/u MAX 30 pts)
- Falta de README (-20 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Falta de orden (entre -5 y -20 pts dependiendo de que tan desordenado)
- Día de atraso (-1 pt por cada hora de atraso)
- Mal nombre en algún archivo entregado (-5 pts c/u)