INF-253 Lenguajes de Programación
Tarea 1: Python

21 de marzo de 2023

1. Código formateado es mejor código

Una programadora competitiva estaba resolviendo problemas. Cuando ella tiene una duda respecto a cómo se resuelve un ejercicio, comienza a leer la solución del problema. El problema es que los programadores competitivos a veces no siguen buenas prácticas para programar, lo que hace el código bastante difícil de leer. A ella le facilita leer un código que siga cierto formato, por esta razón se le ocurrió la increíble idea de hacer un **Formatter**. Este programa consiste en recibir un código fuente junto a una configuración de formato y debe entregar el código formateado.

Se debe utilizar Python 3 y la librería RegEx para las expresiones regulares de la tarea, en caso de que alguna de estas dos condiciones no se cumpla no se revisará la tarea.

2. C−−

La programadora competitiva programa en su lenguaje favorito C−−. La sintaxis de este lenguaje incluye:

- **Declaración de variables**:
  - Los tipos de variables que hay son:
    - **Enteros**: utilizando la palabra `int`, corresponde a cualquier número que no comience con un 0.
    - **Booleanos**: utilizando la palabra `bool`, corresponde a los valores `true` y `false`.
    - **Strings**: utilizando la palabra `str`, corresponde a cualquier secuencia de caracteres entre dos símbolos `#`. Para considerar una variable correctamente declarada debe primero escribirse el tipo y luego el nombre de la variable. El nombre de la variable debe comenzar con una letra y luego le puede seguir tanto letras como números.
    - Ejemplos
      - Correcto: `int var`
      - Incorrecto: `int 1var`
      - Incorrecto: `var`
- **Operaciones Unarias**:
  - **Igualdad**: esta operación se encuentra definida por el símbolo `=`, corresponde a que una variable se le puede entregar el valor de una variable, número, booleano, string u operación.

- **Operaciones Binarias**:
  - **Suma**: corresponde a la suma entre una variable, número, booleano, string u operación con otra utilizando el símbolo `+`.
  - **Resta**: corresponde a la resta entre una variable, número, booleano, string u operación con otra utilizando el símbolo `-`.
  - **Multiplicación**: corresponde a la multiplicación entre una variable, número, booleano, string u operación con otra utilizando el símbolo `*`.
  - **División**: corresponde a la división entre una variable, número, booleano, string u operación con otra utilizando el símbolo `/`.
  - **Mayor que**: corresponde a una comparación de que si una variable, número, booleano, string u operación es mayor que otra utilizando el símbolo `<`.
  - **Igual que**: corresponde a una comparación de igualdad entre una variable, número, booleano, string u operación con otra utilizando el símbolo `==`.
  - Ejemplos:
    - Correctos:
      - `var + var`
      - `var - 1231`
      - `123 * var`
      - `123 / 11`
      - `#aa# + var`
      - `#aa# == var`
      - `123 < #Hola#`
    - Incorrectos
      - `var +`
      - `1ar - 1231`
      - `123 * 1var`
      - `/ 11`
      - `ab# + var`
      - `#ab# <= var`
- **Condicionales**:
  - Los condicionales en este lenguaje corresponde a un `if` con `else`. Escrito de la siguiente forma:

    ```
    if(condicion) {
      codigo
    } else {
      codigo
    }
    ```

    La condición debe corresponder a la operación **Mayor que** o **Igual que**. Siempre debe existir tanto la `if` como `else`, al igual que las llaves.
- **Ciclos**:
  - Los ciclos en este lenguajes corresponde se utiliza la palabra `while`. Escrito de la siguiente forma:

```
while(condicion) {
  codigo
}
```

La condición debe corresponder a la operación **Mayor que** o **Igual que**. Siempre debe existir la palabra `while`, al igual que las llaves.

- Este lenguaje utiliza el símbolo `;` para separara las líneas del código.
- Este lenguaje siempre debe incluir una función sin parámetros llamada `main` que contenga adentro el código.

```
int main() {
  <codigo>
  return 0;
}
```

2.1. EBNF General

```
digito ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

digito_o_zero ::= digito | '0'

variable ::= (A-Z | a-z) { A-Z | a-z | digito_o_zero }

entero  ::= digito {digito_o_zero}

booleano ::= 'true'| 'false'

string ::= '#' { A-Z | a-z | digito_o_zero } '#'

espacio ::= { ' ' | '\t' |'\n' }

oper_bin ::=  (oper_bin | entero | booleano | string | variable)
              espacio
              ('+' | '-' | '/' | '*' | '<' | '==')
              espacio
              (oper_bin | entero | booleano | string | variable)

condicion ::= (condicion | entero | booleano | string | variable)
              espacio
              ('<' | '==')
              espacio
              (condicion | entero | booleano | string | variable)

tipo ::= 'int' | 'bool | 'str'

declaracion ::= tipo espacio variable espacio ';'

igual ::= variable espacio '=' espacio oper_bin espacio ';'
```

```
linea ::= declaracion | igual | condicional | ciclo

bloque ::= '{' { espacio linea } espacio '}'

condicional ::= 'if' espacio '(' espacio condicion espacio ')' espacio
                bloque espacio
                'else' espacio bloque

ciclo ::= 'while' espacio '(' espacio condicion espacio ')' espacio bloque

main ::= 'int' espacio 'main()' espacio
         '{' {espacio linea} espacio
         'return' espacio '0' espacio ';' espacio '}'
```

Cabe destacar que la cantidad de espacios entre palabras puede ser indefinidamente grande.

Ejemplo correcto:

```
int main()           {
str var; int var2;
   bool var3; while(1     < 2) {
var2 = 12312 + #asdasd#; var3 = 1 - 2;if        (sdasd == 3) {}else{int      var1;}
}return 0;}
```

Ejemplo incorrecto:

```
int main()           {
str var; int var2
   bool var3; while(1) {
var2 = 12312 + #asdasd#; var3 = 1 - 2;if(sdasd == 3)else{int      var1;}
}}
```

La razones de porque este código esta mal escrito son:

- `int var2` le falta `;` al final.
- El ciclo no tiene la condición bien escrita.
- El condicional le falta el primer abre y cierra llave.
- No existe el `return 0;`

3. Format

Para que la programadora competitiva pueda leer bien el código la configuración del formato puede recibir los siguientes parámetros:

- **cantEspacios**: corresponde a la cantidad de espacios entre cada secuencia de caracteres.
- **cantSaltosLinea**: corresponde a la cantidad de saltos de línea entre cada linea, estos se deben poner:
  - Después de `int main() {`
  - Después de `return 0;`
  - Después de cada `linea`
  - Después de cada `bloque`
- **cantTab**: corresponde a la cantidad de tabs que debe contener cada línea, esta cantidad depende de cantidad de abre corchete sin cerrar que tiene la línea correspondiente antes.

4. Formatter.py

Usted debe realizar un código en python que lea dos archivos txt llamados `config.txt` y `codigo.txt`, el cual contiene la configuración del formato descrita y el código a formatear. Se deberá formatear el código según la configuración entregada guardando el código en un archivo llamado `formateado.txt`. En caso de que el código entregado este incorrecto, se deberá formatear hasta el primer error encontrado. La configuración en `config.txt` corresponderá a tres enteros en la misma línea y en el mismo orden en la sección 3.

- Ejemplo 1:
  - `config.txt`

    ```
    1 2 1
    ```

  - `codigo.txt`

    ```
    int main()           {
    str var; int var2;
       bool var3; while(1     < 2) {
    var2 = 12312 + #asdasd#; var3 = 1 - 2;if        (sdasd == 3) {}else{int      var1;}
    }return 0;}
    ```

  - `formateado.txt`

    ```
    int main() {

        str var ;

        int var2 ;

        bool var3 ;

        while ( 1 < 2 ) {

            var2 = 12312 + #asdasd# ;

            var3 = 1 - 2 ;

            if ( sdasd == 3 ) {

            } else {

                int var1 ;

            }

        }

        return 0 ;

    }
    ```

- Ejemplo 2:
  - `config.txt`

```
1 2 1
```

- codigo.txt

```
int main()           {
str var; int var2;
   bool var3; while(1     < 2) {
var2 = 12312 + #asdasd#; var3 = 1 - 2;else{int      var1;}
}return 0;}
```

- formateado.txt

```
int main() {

    str var ;

    int var2 ;

    bool var3 ;

    while ( 1 < 2 ) {

        var2 = 12312 + #asdasd# ;

        var3 = 1 - 2 ;
```

5. Datos de Vital Importancia

- Para el tamaño del tab utilice el símbolo espacial de python `\t`.
- La cantidad de ciclos y condicionales anidadas es indefinida.
- Al no hacer uso de expresiones regulares, la tarea no será revisada.

6. Sobre Entrega

- Se deberá entregar un programa llamado Formatter.py.
- Si no existe orden en el código habrá descuento.
- Las funciones implementadas deben ser comentadas de la siguiente forma. **SE HARÁN DESCUENTOS POR FUNCIÓN NO COMENTADA**
  ”’
  Nombre de la función

  Parametro 1 : Tipo
  Parametro 2 : Tipo
  Parametro 3 : Tipo
  .....

  Breve descripción de lo que realiza la función y lo que retorna ”’
- 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.**
- La entrega será vía aula y el plazo máximo de entrega es hasta el **31 de marzo a las 23:55 hora aula**.
- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro la primera hora).
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Solo se pueden realizar consultas respecto a la tarea hasta 2 días antes de la entrega.

7. Calificación

7.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimiento mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

7.1.1. Entrega Mínima

- Crea diferentes expresiones y las utiliza de manera correcta, aprovechándose de la modularización generada.
- Realiza el formateo del código únicamente para sentencias básicas (función main, declaración de variable, operaciones unarias y operaciones binarias) utilizando una única configuración. Ejemplos de sentencias básicas que pueden aparecer en el código: `a + b`, `a * 123123`, etcétera.

7.1.2. Entrega

- Interrupción de la ejecución (total 25 pts):
  1. No detecta correctamente ningún tipo de error (0 pts)
  2. Detecta errores parcialmente interrumpiendo en errores de sintaxis básica, deteniendo la realización del formateo al código (MAX 7 pts)
  3. Detecta errores parcialmente interrumpiendo en errores de sintaxis de mayor dificultada, deteniendo la realización del formateo al código (MAX 15 pts)
  4. Detecta todos los errores correctamente, deteniendo la realización del formateo del código en el momento adecuado (MAX 25 pts)
- Formateo (total 45 pts):
  1. Realiza lo de la entrega mínima (MAX 0 pts)
  2. Realiza el formateo del código únicamente para sentencias básicas (de función main, declaración de variable, operaciones unarias y operaciones binarias) utilizando diferentes configuraciones (MAX 5 pts)
  3. Realiza el formateo del código correctamente para sentencias de mayor dificultad utilizando diferentes configuraciones (MAX 15 pts)

4. Realiza el formateo del código correctamente para sentencias de mayor dificultad permitiendo condicionales o ciclos anidades utilizando diferentes configuraciones (MAX 25 pts)
5. Realiza el formateo del código correctamente para sentencias de mayor dificultad permitiendo condicionales y ciclos anidades utilizando diferentes configuraciones (MAX 35 pts)
6. Realiza el formateo del código correctamente utilizando diferentes configuraciones (MAX 45 pts)

- **NOTA**: Puede existir puntaje parcial, por ejemplo en expresiones regulares puede obtener 3 pts

7.2. 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 (-20 pts)
- Día de atraso (-20 pts por día, -10 dentro de la primera hora)
- Mal nombre en algún archivo entregado (-5 pts c/u)