```
collect to mirror
 peration == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
  _operation == "MIRROR_Y"
"irror_mod.use_x = False
 mirror_mod.use_y = True
 Mrror_mod.use_z = False
  operation == "MIRROR Z"
  rror_mod.use_x = False
  rror mod.use_y = False
  _rror_mod.use_z = True
  election at the end -add
  ob.select= 1
   er ob.select=1
   ntext.scene.objects.acti
   "Selected" + str(modifies
   irror ob.select = 0
    bpy.context.selected_ob
   ata.objects[one.name].sel
   int("please select exact)
     OPERATOR CLASSES ----
   vpes.Operator):
   X mirror to the select
   ject.mirror_mirror_x"
  Fror X"
                     . not
```

LINGUAGEM DE PROGRAMAÇÃO

Natália Queiroz Meezes Carreras

OBJETIVOS:

- Criar uma Linguagem de Programação
- A linguagem deve ter todas as estruturas básicas de uma linguagem de programação: variáveis, condicionais e loops.

MOTIVAÇÃO:

- A razão de ter escolhido esse tipo de linguagem foi misturar duas linguagens que eu acho interessante, Python e C++.
- Gosto da praticidade do C++ assim como a facilidade do Python em relação às faltas de regras.
- Implementei uma linguagem que segue a estrutura de C++, porém sem a necessidade de ponto e vírgula no final das frases..

CARACTERÍSTICAS:

• Estrutura parecida com o C++ porém sem ponto e virgula ao final das frases como representado na EBNF a seguir:

```
BLOCK = { STATEMENT };

STATEMENT = ( λ | ASSIGNMENT | PRINT | WHILELOOP | IFSTATEMENT ), "\n";

ASSIGNMENT = IDENTIFIER, "=", EXPRESSION ;

PRINT = "print", "(", EXPRESSION, ")";

WHILELOOP = "while", "(", EXPRESSION, ")", "{", STATEMENT, "}";

IFSTATEMENT = "if", EXPRESSION, "{", STATEMENT, "}" | "if", EXPRESSION, "{", STATEMENT, "}", "ifnot", "{", STATEMENT, "}";

CONDITION = EXPRESSION, (">" | "<" | ">=" | "<=" | "==" | "!="), EXPRESSION;

EXPRESSION = TERM, { ("+" | "-"), FACTOR };

FACTOR = (("+" | "-"), FACTOR );

FACTOR = (("+" | "-"), FACTOR ) | NUMBER | "(", EXPRESSION, ")" | IDENTIFIER;

IDENTIFIER = LETTER, { LETTER | DIGIT | "_" };

NUMBER = DIGIT, { DIGIT };

LETTER = ( a | ... | Z | A | ... | Z );

DIGIT = ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 );
```

CURIOSIDADES:

- Não precisa colocar ";" no final de cada uma das frases.
- Os nomes das coisas são bem diretos e não causa muita confusão.

EXEMPLOS:

Aqui está um exemplo de como será a implementação de um código na linguagem.

```
# Test File
# Variable Declaration and Assignment
int x = 10
double y = 5.5
# Function Declaration
int add(int a, int b) {
    return a + b
# Function Call
int result = add(x, 20)
# Print Statement
print(result)
# If Statement
if (result > 15) {
    print("Result é maior que 15")
} else {
    print("Resultado não é maior que 15");
# While Loop
int i = 0
while (i < 5) {
    print(i)
   i = i + 1
```

12/10/2023 Sample Footer Text 6

FONTES:

- Toy, Own. "Writing Your Own Toy Compiler Using Flex, Bison and LLVM." *Gnuu.org*, 18 Sept. 2009, gnuu.org/2009/09/18/writing-your-own-toy-compiler/. Accessed 10 Dec. 2023.
- "Practical Parsing with Flex and Bison." Begriffs.com, 2021, begriffs.com/posts/2021-11-28-practical-parsing.html. Accessed 10 Dec. 2023.
- Hughes, Wilfred. "Baby Steps to a c Compiler." Wilfred Hughes::Blog, 2014, www.wilfred.me.uk/blog/2014/08/27/baby-steps-to-a-c-compiler/. Accessed 10 Dec. 2023.