DESCOMP

É a linguagem descomplicada ideal para o seu primeiro contato com a programação.

Juan Jorge Garcia

MOTIVAÇÃO

Atualmente, nota-se um grande crescimento do mercado de tecnologia no Brasil e no Mundo, de 95% no número de postos de trabalho nos últimos 12 anos. Além disso, todos os anos vê-se grandes aumentos no valor investido nesse setor.

Entretanto, nota-se ainda uma grande dificuldade para encontrar profissionais especializados na área. Isso deve-se ao fato de que aprender a programar não é uma tarefa simples. De acordo com pesquisas, apenas 1% da população brasileira fala inglês fluente, sendo isso um dos principais agravantes no aprendizado da programação.

Afim de democratizar o acesso e o aprendizado da tecnologia para a população, foi criado a linguagem Descomp.

Fontes: https://canaltech.com.br/empregos/quer-um-emprego-em-ti-veja-quanto-a-area-cresceu-no-brasil-nos-ultimos-anos-173224/ e https://www.moneytimes.com.br/apesar-de-estar-no-curriculo-apenas-1-dos-brasileiros-realmente-fala-ingles-fluente/

DESCOMP

Descomp é uma linguagem de programação em português baseada nas linguagens Julia, C e Python.

O objetivo é ser uma linguagem simples e natural para o primeiro contato do usuário com a programação. A sintaxe é construída de uma forma que se assemelha a linguagem natural do programador brasileiro, trazendo, assim, um maior entendimento.

CARACTERÍSTICAS

Apresenta as estruturas básicas de uma linguagem de programação:

- Variáveis
- Condicionais
- Loops
- Funções
- Operações binarias
- Operações unárias
- Tipos: Inteiros, booleanos e strings.

EBNF

```
* PROGRAM = { FUNCTION | COMMAND } ;
* BLOCK = { COMMAND } ;
* FUNCTION = "func", IDENTIFIER, "(", (TYPE, IDENTIFIER ), {"," , TYPE, IDENTIFIER}, ")", "->", TYPE, "\n", BLOCK, "fim" ;
* FUNCALL = IDENTIFIER, "(", (REL_EXPRESSION), {"," , REL_EXPRESSION}, ")" ;
* COMMAND = ( λ | ASSIGNMENT | PRINT | IF | WHILE | LOCAL | RETURN | FUNCALL), "\n" ;
* RETURN = "retorne", REL_EXPRESSION ;
* LOCAL = TYPE, IDENTIFIER;
* ASSIGNMENT = IDENTIFIER, "=", REL_EXPRESSION | "ler_entrada", "(", ")" ;
* PRINT = "mostre", "(", REL_EXPRESSION, ")" ;
* EXPRESSION = TERM, { ("+" | "-" | "||"), TERM } ;
* REL_EXPRESSION = EXPRESSION, { ("==" | ">" | "<"), EXPRESSION };
* WHILE = "enquanto", REL_EXPRESSION, "\n", BLOCK, "fim";
* IF = "se", REL_EXPRESSION, "\n", BLOCK, { ELSEIF | ELSE }, "fim";
* ELSEIF = "senao_se", REL_EXPRESSION, "\n", BLOCK, { ELSEIF | ELSE };
* ELSE = "senao", "\n", BLOCK;
* TERM = FACTOR, { ("*" | "/" | "&&"), FACTOR } ;
* FACTOR = (("+" | "-" | "!"), FACTOR) | NUMBER | BOOLEAN | STRING | "(", REL_EXPRESSION, ")" | IDENTIFIER | FUNCALL;
* IDENTIFIER = LETTER, { LETTER | DIGIT | "_" } ;
* TYPE = "int" | "bool" | "str";
* NUMBER = DIGIT, { DIGIT } ;
* STRING = '"', {.*?}, '"';
* BOOLEAN = "true" | "false";
* LETTER = ( a | ... | z | A | ... | Z ) ;
* DIGIT = ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 ) ;
```

EXEMPLOS DE USO

Funções e recursão

```
func soma(int x) -> int #= declaracao de funcao: func identifier(tipo arg1) -> tipo=#
    int test #= declaracao de variavel: tipo identifier =#
    int a

    a = x + 1 #= atribuicao =#
    mostre(a) #= print =#
    se a < 5 #= inicio da condicional =#
        a = soma(a)
    fim  #= fim da condicional =#
    retorne a #= return =#
fim #= fim da funcao =#
int a
a = 1
a = soma(a)
mostre(a)
```

Código

```
/m/c/U/j/7/1/compiler (master±) ▶  python3.8 main.py ./testes/teste1.jj
2
3
4
5
5
```

Resultado

EXEMPLOS DE USO

Condicionais e loop

```
int x  #= declaracao de variavel =#
int x1
int y2
str j
bool verdade
int y
x = 5 #= atribuicao de variavel =#
x1 = 0
y2 = 10
y = ler_entrada() #= input =#
j = "oi teste"
mostre(j) #= print =#
verdade = true
se (verdade) #= inicio da condicional =#
    mostre("variavel booleana funciona")
fim #= fim da condicional =#
enquanto (x1 < y) || (x1 == y) #= loop (while) =#
    se x1 < 3
        mostre(2)
    senao_se x1 == 3 #= elseif =#
        mostre(20)
    senao_se x1 == 4
        mostre(200)
    senao #= else =#
        mostre(2000)
    fim

    se (!(x1 > 3) && (y2 == 10))
        mostre(10)
        mostre("dale")
    senao
        mostre(100)
    fim

    mostre((x1 > 3) && (x1 < y2))
    mostre(x1)
    x1 = x1 + 1
fim
```

Código

```
/m/c/U/j/7/1/compiler (master±) ▶  python3.8 main.py ./testes/teste2.jj
5
oi teste
variavel booleana funciona
2
10
dale
False
0
2
10
dale
False
1
2
10
dale
False
2
20
10
dale
False
3
200
100
True
4
2000
100
True
5
```

Resultado

EXEMPLOS DE USO

Operações de tipos

```
mostre("operacoes de tipos")
mostre(1 + true)  #= Ok =#
mostre("ok")
mostre(1 && true) #= Ok =#
mostre("ok")

mostre("a" * 1 * true) #= Ok =#
mostre("ok")

mostre("a" == "b") #= Ok, resultado bool: False =#
mostre("ok")
```

Código

```
/m/c/U/j/7/1/compiler (master±) ▶  python3.8 main.py ./testes/teste3.jj
operacoes de tipos
2
ok
True
ok
a1True
ok
False
ok
```

Resultado

OBRIGADO