

# Motivação

- Meu primeiro jogo desenvolvido na disciplina Jogos Digitais teve como inspiração o Inferno de Dante, da obra "A Divina Comédia".
- Assim, em uma tentativa em vão de redimir meus pecados, por que não aproveitar a disciplina de Lógica da Computação para desenvolver uma linguagem de cunho celestial?





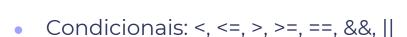
## Casos de uso

- Ainda existem comunidades no mundo aversas aos avanços tecnológicos do último século. Uma das mais conhecidas são os Amish, um grupo religioso cristão.
- Por ter uma sintaxe e temática mais alinhada com os interesses dessa comunidade, a JavaScripture pode ser a porta de entrada das novas gerações de Amish para o mundo moderno.





# Features Principais



- UnOps: +, -,!
- BinOps: +, -, \*, /, .,
- While loop
- Print
- If / Else
- Functions
- Comentários (com #)





## Desenvolvimento

- A linguagem foi desenvolvida com base na versão 2.4 do compilador desenvolvido no decorrer da disciplina.
- A linguagem tem todas as estruturas básicas de uma linguagem de programação: variáveis, condicionais, loops e funções. No entanto, ela é limita em alguns aspectos. Por exemplo, os loops só podem ser feitos com o uso do while.
- Ela se assemelha a linguagem C em diversos aspectos, com exceção das reserved keywords e outras particularidades que serão desenvolvidas a seguir.

# C vs JavaScipture

C

```
int num1 = 5;
int num2 = 8;
int sum = 0;

while (sum <= 20) {
  sum = num1 + num2;
  num1++;
  num2++;
}

printf("Sum is greater than 20.
  Exiting the loop.");</pre>
```

### JavaScripture

```
baptize int as num1 giveth 5;
baptize int as num2 giveth 8;
baptize int as sum giveth 0;

prayAsLongAs (sum <= 20) {
  sum giveth num1 + num2;
  num1 giveth num1 + 1;
  num2 giveth num2 + 1;
  };

proclaim("Sum is greater than 20.
  Exiting the loop.");
```





baptize int as x1; baptize int as x2 giveth 42; baptize string as s1 giveth "Hello World!";

### Variable Assignment

xl giveth 2; sl giveth "Good morning!";

### Sintaxe

### Loop (While)

```
baptize int as x1 giveth 0;
prayAsLongAs (x1 < 5) {
    x1 giveth x1 + 1;
};</pre>
```

#### **Function Declaration and Call**

```
preach int as somaUm(int as f_x1) {
    f_x1 giveth f_x1 + 1;
    amen f_x1;
};
baptize int as x1 giveth 0;
x1 giveth somaUm(x1);
```

Obs: note a necessidade de ; ao final de qualquer statement (logo após o bracket de fechamento).

## Sintaxe





#### Tf

```
baptize int as x1 giveth 0;
testify (x1 == 0) {
    x1 giveth 10;
};
```

#### If and Else

```
baptize int as x1 giveth 0;
testify (x1 == 0) {
    x1 giveth 10;
} otherwise {
    x1 giveth 5;
};
```



Obs: como dito anteriormente, um ; é necessário após cada Statement. No entanto, um if else conta como apenas um statement. Por esse motivo não é necessário ; antes do otherwise.

## Sintaxe



baptize string as s1 giveth "Hello"; baptize string as s2 giveth "World!"; proclaim(s1.s2);

#### **Comments**

baptize string as s1 giveth "Hello";
# baptize string as s2 giveth " World!";
proclaim(s1.s2);



O Código acima produzirá um erro, pois s2 não foi declarada



- Como foi desenvolvida a partir do compilador de sala de aula, o interpretador de JavaScripture pode ser exetutado a partir da pasta conceitoB com:
- Python3 main.py input.amen
- O interpretador somente aceitará arquivos sagrados com a extensão .amen .



- Comando:
- python3 main.py tests/shouldSucceed/test5.amen

```
# Contéudo de test5.amen:
baptize string as sufixo giveth ", amen!";
baptize string as prefixo giveth "There shall be no DP";
baptize string as frase giveth prefixo.sufixo;
baptize int as i;
i giveth 0;
prayAsLongAs((i <= 10)) {
proclaim(i.": ".frase);
i giveth i + 1;
```

```
~/Documents/logcomp/JavaScripture/conceitoB git:(mai python3 main.py tests/shouldSucceed/test5.amen

0: There shall be no DP, amen!
1: There shall be no DP, amen!
2: There shall be no DP, amen!
3: There shall be no DP, amen!
4: There shall be no DP, amen!
5: There shall be no DP, amen!
6: There shall be no DP, amen!
7: There shall be no DP, amen!
9: There shall be no DP, amen!
10: There shall be no DP, amen!
```



- Foi desenvolido um programa para realizar os testes automatizados. Dentro da pasta tests, existem dois diretórios: shouldFail e shouldSucceed.
- O script de testes executará e comparará o resultado com a devida pasta.
- Para executar, execute:
- Python3 run\_tests.py

```
~/Documents/logcomp/JavaScripture/conceitoB git:(main) (0.416s)
python3 run tests.py
Test passed: tests/shouldSucceed/test2.amen
Test passed: tests/shouldSucceed/test3.amen
Test passed: tests/shouldSucceed/test4.amen
Test passed: tests/shouldSucceed/test5.amen
Test passed: tests/shouldSucceed/test6.amen
Test passed: tests/shouldSucceed/test0.amen
Test passed: tests/shouldSucceed/test1.amen
Test passed (expected failure): tests/shouldFail/test2.amen
Test passed (expected failure): tests/shouldFail/test3.amen
Test passed (expected failure): tests/shouldFail/test4.amen
Test passed (expected failure): tests/shouldFail/test5.amen
Test passed (expected failure): tests/shouldFail/test6.amen
Test passed (expected failure): tests/shouldFail/test0.amen
Test passed (expected failure): tests/shouldFail/test1.amen
```



## Curiosidades

- Outra inspiração para o projeto foi o existente TempleOS.
- Trata-se de um sistema operacional escrito em HolyC, uma espécie de versão celestial do C desenvolvida especificamente para desenvolver o TempleOS.



