FUNDAÇÃO EDSON QUEIROZ

UNIVERSIDADE DE FORTALEZA

CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT

CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DISCIPLINA: PROGRAMAÇÃO FUNCIONAL

RELATÓRIO FINAL

FORTALEZA - CE
2024

Papéis dos integrantes

Requisitos: Adberto Melo - 2213938
Codificação: Adberto Melo - 2213938
Testes: Adberto Melo - 2213938

Obs.: Nenhum aluno quis aceitar o desafio de aprendizagem de tentar desenvolver um pequeno transpilador

Introdução

O projeto desenvolvido foi baseado na primeira sugestão de projeto do documento “SUGESTÕES DE IDEIAS DE PROJETOS_N704_PROGRAMAÇÃO FUNCIONAL_ATIVIDADE FINAL_2024.1A.pdf”. No documento pede-se para que seja desenvolvido um “Tradutor de código (dois grupos com linguagens de entrada e saída diferentes)”.

Foi desenvolvido um “transpilador” de JavaScript para Python, que atende aos requisitos propostos no documento, que são: “declaração, implementação e chamada de funções”,”if-then-elses e for/while”,”entradas e saídas pelo terminal”,”instruções de atribuição”, “expressões numéricas e booleanas”.

Foi usado um código-fonte base em JavaScript com os requisitos acima e a sua consequente conversão em Python. Como suíte para testes foi usado o Jest.

Requisitos

Não Funcionais

- o programa deve permitir a execução via linha de comando
- o programa deve “transpilar” o código de forma eficiente
- o programa deve gerar o código em Python que seja executável
- o programa deve atender aos requisitos da proposta 1 do documento sugestões de projeto disponibilizado pelo professor
- o programa deve ter testes que atendam aos requisitos propostos no documento de sugestões de projeto(projeto 1)

Funcionais

- o programa deve permitir a conversão de um código Javascript para Python(**function transpiler**);
- o programa deve permitir a conversão de uma declaração de função(**function variableDeclaration**);
- o programa deve permitir a conversão da implementação de uma função(**function callExpressionFn**);
- o programa deve permitir a conversão de uma chamada de função(**function callExpressionFn**);
- o programa deve permitir a conversão de uma estrutura de controle if-else(**function ifStatement**);
- o programa deve permitir a conversão de um loop while(**function whileStatement**);
- o programa deve permitir a conversão de uma entrada via command line(**function getCalleeName**);
- o programa deve permitir a conversão de uma saída via command line(**function getCalleeName**);
- o programa deve permitir a conversão de uma instrução de atribuição(**function expressionStatement**);
- o programa deve permitir a conversão de uma expressão numérica(**function assignmentExpression**);
- o programa deve permitir a conversão de uma expressão booleana(**function getBinaryExpression**);

Execução do programa

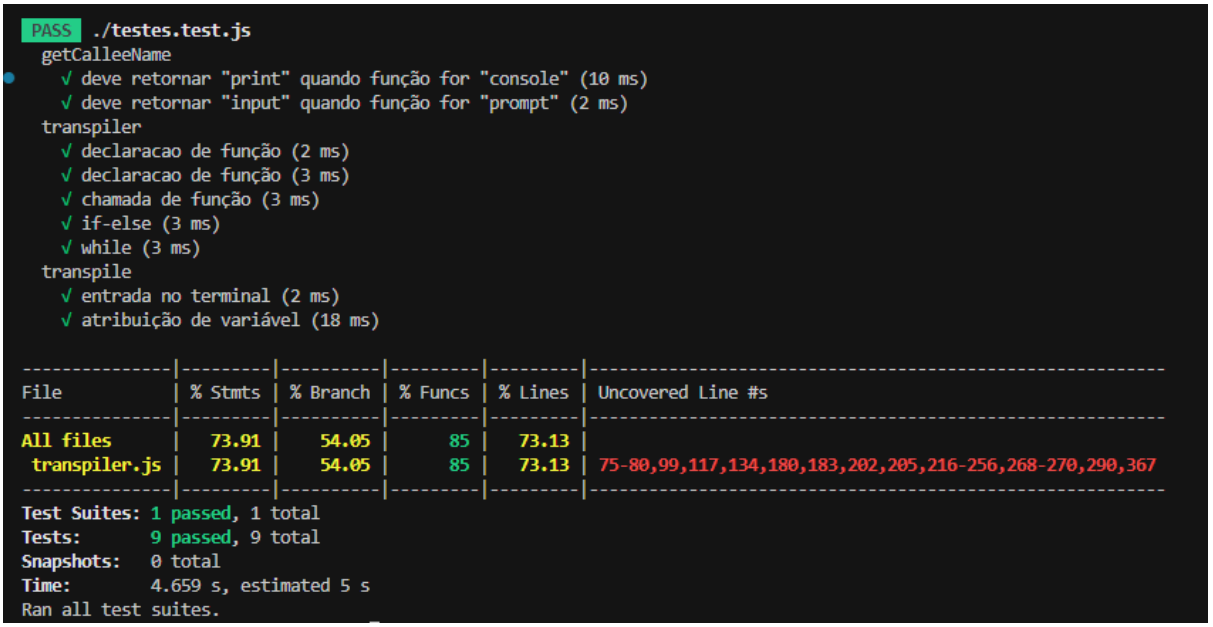
O código-base usado para a “transpilação” foi o código abaixo. O código solicita um número ao usuário, se o número for maior que zero, é impresso no terminal a sequência de números.

```
function printValues(value)
{
   if (value <= 0)
   {
     console.log('Digite um valor maior que zero');
     return;
   }

   while(value > 0)
   {
     console.log(value);
     value = value - 1;
   }
}

var value = Number(prompt('Digite um valor'));

printValues(value);
```

O resultado da “transpilação” foi:

```
PS C:\Dev\Testes\transpiler\acorn> node transpiler.js
def printValues(value):
  if value <= 0:
    print('Digite um valor maior que zero')
    return
  while value > 0:
    print(value)
    value=value-1
value=(int)(input('Digite um valor'))
printValues(value)
```

Resultado do code coverage



```
PASS ./testes.test.js
  getCalleeName
    √ deve retornar "print" quando função for "console" (10 ms)
    √ deve retornar "input" quando função for "prompt" (2 ms)
  transpiler
    √ declaracao de função (2 ms)
    √ declaracao de função (3 ms)
    √ chamada de função (3 ms)
    √ if-else (3 ms)
    √ while (3 ms)
  transpile
    √ entrada no terminal (2 ms)
    √ atribuição de variável (18 ms)

---------------|---------|----------|---------|---------|-------------------------------------------------------------
File           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------|---------|----------|---------|---------|-------------------------------------------------------------
All files      |   73.91 |    54.05 |      85 |   73.13 |
 transpiler.js |   73.91 |    54.05 |      85 |   73.13 | 75-80,99,117,134,180,183,202,205,216-256,268-270,290,367
---------------|---------|----------|---------|---------|-------------------------------------------------------------
Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        4.659 s, estimated 5 s
Ran all test suites.
```

Pontos do código implementação do paradigma funcional

1) Função/Declaração Lambda

Foram definidas algumas funções lambdas tais como a da linha 31

```
const identacao = (c) => {
```

2) List comprehension

O mais próximo em JavaScript de uma list comprehension é um “filter” em um array. Não encontrei nenhum ponto do código em que eu pudesse aplicar um filter sem aumentar a complexidade do código;

3) Função de continuação

Também não encontrei nenhum ponto em que eu pudesse aplicar uma função de continuação;

4) Closure

Apliquei closure na função “indentacao”, linha 31. Essa função é que vai permitir ao usuário escolher o tipo de indentação do código, como, por exemplo, “tab” ou espaços.

5) Função de alta ordem

Uso uma função de alta ordem na função "transpilerTree”, já que ela recebe a função de indentação como parâmetro

6) Monad

O mais próximo de uma monad foi a criação de um bloco try-catch. Inseri esse bloco na linha 361. Nesse ponto, se o código javascript estiver mal-formado, será capturado um erro.

Repositório github

https://github.com/adbertomelo/unifor_programacao_funcional.git

Conclusão

O processo de “transpilação” de código de uma linguagem de programação para outra é complexo. Iniciei esse projeto achando que não iria conseguir e apesar de ser um “transpilador” bem rudimentar, foi um grande aprendizado sobre técnicas de “transpilação” de código e sobre técnicas de programação funcional.