éticos19 ógicos39 ̧ão49 ̧ão510 ência710

GH - Uma Linguagem de Exploração de Dados no GitHub

Caio Barbosa

July 2018

Capítulo 1

1 Introdução

1.1 Definição da Linguagem

A Linguagem GH tem como objetivo facilitar a vida de pesquisadores que fazem Mineração de Dados no GitHub. A Linguagem permite a fácil manipulação dos tipos da API, transformando-os em objetos manipuláveis em uma linguagem de programação.

1.2 Informações Gerais

- Um programa na linguagem GH é composto por funções. A função inicial do programa é a função ***main***.

```
void main(){
    comandos;
}
```

- Os comentários são permitidos, sendo *inline*, o compilador ignora tudo na linha após os caracteres '/#'.

1.3 Variáveis

Os escopos das variáveis na linguagem GH são locais, existindo apenas dentro da função em que foram declarados. O escopo das funções é delimitado pelos símbolos '{' e '}'. E o terminador de comandos é o símbolo ';'

As variáveis são nomeadas através de identificadores (nomes), iniciados por uma letra, maiúscula ou minúscula (visto que a linguagem não é *case-sensitive*), podendo ser seguidos de letras, números ou o símbolo *underscore* ('_') a partir do segundo caracter.

Os nomes podem ter no máximo 16 caracteres, e apenas admitem o underscore ('_') como símbolo.

As palavras reservadas são nomes exclusivos para os Tipos (1.6), Operadores (2.0), Instruções (3.0) e Funções da Linguagem (4.0).

A declaração das variáveis é feita utilizand o símbolo **<-**, como exemplo:

```
int num <- 2;
```

O acesso aos campos das variáveis (tipos da API (1.6)) é feito através do símbolo **->**, como exemplo:

```
commit1 -> id;
```

1.4 Declaração de Funções

As funções declaradas são de escopo global, não podendo ser operação de nenhum tipo (os tipos são finais).

Sendo sua sintaxe de declaração:

```
<retorno> <nome da func> (<tipo1> <nome_do_param1>, ...){
    comandos;

    return <variavel do tipo do retorno>
}
```

A sintaxe de chamada das funções declaradas pelo usuários:

- Retorno Vazio

```
nome_da_func(param1, param2);
```

- Retorno com tipo

```
<tipo1> var <- nome_da_func(param1, param2);
```

1.5 Coerção

A linguagem GH é fortemente tipada, porém provê coerção, forçando todos os tipos a virarem o tipo string, de maneira automática, ao serem concatenados à uma string com o operador de concatenação *+=*.

1.6 Tipos de Dados:

1.6.1 bool

Descrição:

Possui dois valores representados por true e false.
O valor *default*(null) do bool é false.

Operações:

Utilizado em operações de negação, conjunção ou disjunção.

1.6.2 int

Descrição:

Utilizado para armazenar números inteiros com tamanho máximo de 32 bits.

O valor *default*(null) do int é 0.

Operações:

Soma, Subtração, Multiplicação, Divisão e Resto.

1.6.3 float

Descrição:

Utilizado para armazenar números de ponto flutuante. A parte inteira é separada da parte fracionária por um '.'.

O valor *default*(null) do float é 0.0.

Operações:

Soma, Subtração, Multiplicação, Divisão e Resto.

1.6.4 string

Descrição:

Representação de um caractere ou conjunto de caracteres.

Operações:

Comparação através da função ***compare***.

Concatenação através do operador de concatenação (+=).

Converter de maiúscula para minúscula e vice-versa através das funções da linguagem ***upper*** e ***lower***.

Busca de caracteres através da função da linguagem ***find***

Split da string através da função da linguagem ***split***.

Tamanho da string através da função da linguagem ***length***.

1.7 Tipos Associados à GitHub API

Descrição:

Representação em forma de Objeto do JSON retornado pela API[1] para a endpoint do tipo respectivo.

Campos:

Os campos são informações que vem da API para serem manipuladas de maneira read-only. (Não é possível modificar a informação, apenas operações de leitura).

1.7.1 user

Campos:

a) login (string)

[1] https://developer.github.com/v3/

b) id (int)

c) type (string)

d) name (string)

e) company (string)

f) email (string)

g) public-repos (int)

h) followers (int)

i) following (int)

j) created-at (date)

1.7.2 repository

Campos:

a) owner (string)

b) id (int)

c) name (string)

d) full-name (string)

e) description (string)

f) private (boolean)

g) fork (boolean)

h) url (string)

i) language (strings)

j) commit-list (commits)

1.7.3 commit

Campos:

a) sha (string)

b) author-name (string)

c) author-login (string)

d) author-name (string)

e) author-date (string)

f) commiter-name (string)

g) commiter-login (string)

h) commiter-email (string)

i) commiter-date (string)

j) additions (int)

k) deletions (int)

l) file-list (files)

1.7.4 file

Campos:

a) filename (string)

b) additions (int)

c) deletions (int)

d) patch (string)

1.7.5 issue

Campos:

a) id (int)

b) title (string)

c) state (string)

d) url (string)

e) number (int)

f) body (string)

g) author-login (string)

h) comments (comments)

i) created-at (date)

j) closed-at (date)

k) closed-by (string)

1.7.6 pullrequest

Campos:

a) id (int)

b) title (string)

c) state (string)

d) url (string)

e) number (int)

f) body (string)

g) author-login (string)

h) comments (comments)

i) created-at (date)

j) merged-at (date)

k) merged-by (string)

l) merge-commit

m) commits (int)

n) additions (int)

o) deletions (int)

p) changed-files (int)

1.7.7 comment

Campos:

a) url (string)

b) id (int)

c) body (string)

d) position (int)

e) commit-hash (string)

f) author-login (string)

g) created-at (date)

h) updated-at (date)

1.7.8 date

ISO 8601 timestamp com informação de fuso.

Campos:

a) day (string)

b) month (string)

c) year (string)

d) hour (string)

e) minute (string)

f) second (string)

g) timezone (string)

h) date (string)

Operações:

- Comparação através da função ***compare***.
- Atualização (buscar dados na API novamente) através da função ***refresh***.
- Subtração (para o tipo ***date*** através da função ***diff_dates***.

1.8 Listas

As Listas:

- São representados em linguagem natural pelo plural do seu tipo (e.g. strings).
- Nunca são nulos, ao ser declarado, ele é instanciado com tamanho zero, sem necessidade de atribuição.
- Não possuem campos. Para conseguir informações sobre as Listas deve-se utilizar as funções da linguagem, como por exemplo: ***length***.

1.8.1 Operações:

As operações suportadas pela pelas Listas são:

- Ordenação através da função ***sort***.
- Acesso aos elementos de forma única através do ***foreach***
- Adição de elementos através da função ***add***
- Remoção de elementos através da função ***remove***
- Retorno do tamanho através da função ***length***

Table 1: Aritméticos

Operador	**Significado**
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto
-	Unário Negativo

Table 2: Relacionais

Operador	**Significado**
greater	Maior que
less	Menor que
is	Igual a
isnot	Diferente de
greater-equal	Maior ou Igual
less-equal	Menor ou Igual

Table 3: Lógicos

Operador	**Significado**
true	Verdadeiro
false	Falso
not	Negação
and	Conjunção
or	Disjunção

Table 4: Concatenação

Operador	**Significado**
+=	Concatenação

2 Capítulo 3 - Operadores

- A tabela acima lista as regras de precedência e de associatividade dos operadores.
- Operadores na mesma linha tem a mesma ordem de precedência.

Table 5: Atribuição

Operador	**Significado**
<-	Atribuição
- -	Decremento
++	Incremento

Table 6: Acesso aos Campos

Operador	**Significado**
->	Acesso aos Campos

Table 7: Precedência

Operador	**Associatividade**
( )	esquerda para direita
!	esquerda para direita
* / %	esquerda para direita
greater greater-equal less less-equal	esquerda para direita
is inst	esquerda para direita
and	esquerda para direita
or	esquerda para direita
<-	esquerda para direita

- A tabela está em ordem decrescente de precedência (e.g. * e / tem a mesma ordem de precedência, porém precedência maior que + e -).

3 Capítulo 4 - Instruções

3.1 Estrutura Condicional de Uma Via

```
if (<expr>) {
    comando;
}
```

3.2 Estrutura Condicional de Duas Vias

```
if (<expr>) {
    comando;
} else {
    comando;
```

```
}
```

3.3 Estrutura Iterativa

```
foreach <element> in <list> {
    comando;
}
```

3.4 Estrutura Iterativa de Controle Lógico

```
while (<expr>) {
    comando;
}
```

4 Capítulo 5 - Funções da Linguagem

4.1 add

A função ***add*** é uma função para do tipo Lista, e adiciona um elemento à lista.
Sua sintaxe de uso é:

```
lista_de_commits -> add(meu_commit);
```

4.2 remove

A função ***remove*** é uma função para do tipo Lista, e remove um elemento da lista, se o elemento existir.
Sua sintaxe de uso é:

```
lista_de_commits -> remove(meu_commit);
```

4.3 sort

A função ***sort*** é uma função para do tipo Lista, e ordena os elementos de maneira crescente ou decrescente.
- true (crescente)
- false (decrescente)

Sua sintaxe de uso é:

```
lista_de_commits -> sort(true); //(crescente)
lista_de_commits -> sort(false); //(decrescente)
```

4.4 upper

A função ***upper*** é uma função para do tipo string, e passa todos os caracteres da string para maiúsculos.
Sua sintaxe de uso é:

```
minha_string -> upper(); //
```

4.5 collect_data

A função ***collect_data*** é uma função para do tipo repository, ela é responsável por coletar informações do repositório na Github API.

Sua sintaxe de uso é:

```
repository meu_projeto -> collect_data("project_name");
```

4.6 refresh

A função ***refresh*** é uma função dos tipos associados à API, ela é responsável por atualizar os dados (fazer uma nova consulta à API) do tipo que chamar essa função.

Sua sintaxe de uso é:

```
meu_dado_da_api -> refresh();
```

4.7 diff_dates

A função ***diff_dates*** é do tipo dates, e contabiliza a quantidade de dias entre a date atual (variável chamando a função) e a data passada como parâmetro, retornando um inteiro representando a quantidade de dias.

Sua sintaxe de uso é:

```
date hoje <- "24/10/2018";
date amanha <- "25/10/2018";

int days <- hoje -> diff_dates(amanha);
```

4.8 length

A função ***length*** é dos tipos Lista e string, e retorna o tamanho da Lista ou da string.

Sua sintaxe de uso é:

```
int tam <- minha_string -> length();
int tam2 <- minha_lista -> length();
```

4.9 compare

A função ***compare*** serve para todos os tipos. Ela verifica se os elementos comparados (ou listas) são idênticos. No caso das listas, ele compara elemento à elemento, com a ordem importando. Retornando um bool.

Sua sintaxe de uso é:

```
commits meus_commits1 <- meu_repositorio1 -> commit_list;

commits meus_commits2 <- meu_repositorio2 -> commit_list;

bool check <- meus_commits1 -> compare(meus_commits2);
```

O retorno das funções é feito utilizando a palavra reservada ***return*** seguido da variável a ser retornada.

5 Capítulo 6 - Exemplos

```
void main() {
    print("Hello world");
}


void main() {

    string project_name <- "project_name";
    repository github_project -> collect_data(project_name);
    commits project_commits <- github_project -> commit_list;

    foreach commit_1 in project_commits {
        if (commit_1.id is 2) {
            print("This is the second commit of the project: " + project_name);
        }
    }
}
```

6 Capítulo 7 - Tokens

A linguagem Java será utilizada para a implementação dos analisadores léxicos e sintáticos da Linguagem GH.

6.1 Enumeração dos Tokens

```
    public enum GHTokens{

        main, id, tVoid, tInt, tInts, tFloat,
        tFloats, tString, tStrings,
        tBool, tBools, tCommit, tCommits,
        tUser, tUsers, tRepository,
        tRepositories, tFile, tFiles, tIssue,
        tIssues, tPullRequest,
        tPullRequests, tComment, tComments,
```

```
    tDate, tDates, escStart,
    escEnd, paramStart, paramEnd, cmt,
    term, sep1, sep2, prRead,
    prWrite, prIf, prElse, prWhile, prFor,
    prReturn, opAtrib, opIncre,
    opLogicAnd, opLogicOr, opLogicNeg,
    opAritAd, opAritMul, opNegUn,
    opRel1, opRel2, opRel3, opConc;

}
```

6.2 Expressões Regulares Auxiliares

```
letra = '[xyz]';

digito = '[0-9]';
```

6.3 Lexemas:

6.3.1 Main:

```
main = 'main';
```

6.3.2 Identificador:

```
id = ('letra') ('letra' | '_' | 'digito')*;
```

6.3.3 Tipos Primitivos:

```
tVoid = 'void';

tInt = 'int';

tInts = 'ints';

tFloat = 'float';

tFloats = 'floats';

tString = 'string';

tStrings = 'strings';

tBool = 'bool';

tBools = 'bools';

tCommit = 'commit';
```

```
tCommits = 'commits';

tUser = 'user';

tUsers = 'users';

tRepository = 'repository';

tRepositories = 'repositories';

tFile = 'file';

tFiles = 'files';

tIssue = 'issue';

tIssues = 'issues';

tPullRequest = 'pullrequest';

tPullRequests = 'pullrequests';

tComment = 'comment';

tComments = 'comments';

tDate = 'date';

tDates = 'dates';
```

6.4 Delimitadores:

6.4.1 Escopo:

```
escStart = '{';
escEnd = '}';
```

6.4.2 Parâmetros:

```
paramStart = '(';
paramEnd = ')';
```

6.4.3 Comentários:

```
cmt = '/#';
```

6.4.4 Terminador:

```
term = ';'
```

6.4.5 Separador:

```
sep1 = '.';
sep2 = 'in';
```

6.5 Constantes (?):

6.6 Palavras Reservadas:

6.6.1 Entrada e Saida:

```
prRead = 'read';
prWrite = 'print';
```

6.6.2 Iteração ou Seleção:

```
prIf = 'if';
prElse = 'else';
prWhile = 'while';
prFor = 'foreach';
```

6.6.3 Retorno:

```
prReturn = 'return';
```

6.7 Operadores:

6.7.1 Atribuição:

```
opAtrib = '<-';
opIncre = '++' | '--';
```

6.7.2 Lógicos:

```
opLogicAnd = 'and';
opLogicOr = 'or';
opLogicNeg = 'not'
```

6.7.3 Aritméticos:

```
opAritAd = '+' | '-';
opAritMul = '*' | '/' | '%";
```

6.7.4 Unário

```
opNegUn = '-';
```

6.7.5 Relacionais:

```
opRel1 = 'greater'|'less';
opRel2 = 'greater-equal'|'less-equal';
opRel3 = 'is' | 'isnot';
```

6.7.6 Concatenação:

```
opConc = '+=';
```