Metody Translacji
Interpreter języka skryptowego *python-like* z wbudowanymi obiektowymi typami danych *datetime*, *period* oraz *int*
Dokumentacja końcowa

Maciej Salwowski 298904

26 stycznia 2022

1 Opis projektu

W ramach projektu został stworzony interpreter języka Python like, który zamiast wcięć interpretuje nawiasy klamrowe.

2 Opis gramatyki

W tym rozdziale przedstawiona została lista wykorzystywanych tokenów przez interpreter oraz struktura gramatyki opisana notacją EBNF (*Extended Backus-Naur Form*).

2.1 Lista tokenów

Operatory matematyczne oraz znaki:

```
'+', '-', '*', '/', '(', ')', '{', '}', '=', '==', '!=', '>', '>=', '<', '<=', ':',
↪  '.'
```

Słowa kluczowe:

```
"if", "else", "while", "def", "return", "or", "and
```

2.2 Gramatyka

Aby schemat produkcji był możliwie łatwo czytelny zamiast znaku alternatywy każdy z jej argumentów opisany został w oddzielnej linii (w przypadku niemieszczenia się w jednej linii jej zawartość przeniesiona jest do kolejnego nieponumerowanego wiersza).

```
program                                 =   definitions, statements;

definitions                             =   definition, {definition};

definition                              =   "def", identifier, '(', parameters,
↪  ')', ':', block;

parameters                              =   [identifier, {',', identifier}];

assignable                              =   identifier, ['.', identifier];

statements                  =   { statement };

```

```
block                                    =   '{', statements, '}';

statement                                  =        if_statement
                                 while_statement
                                 return_statement
                                 assignable, assignable_statement;

assignable_statement        =   assign_statement
                                 function_call_statement;

if_statement                           =   "if", '(', logic_expression, ')', ':',
↪  block, ["else", ':', block];

while_statement                        =   "while", '(', logic_expression, ')',
↪  block;

return_statement                       =        "return", logic_expression;

function_call_statement     =   assignable, '(', [arguments], ')';

arguments                                =   logic_expression, {',',
↪  logic_expression};

logic_expression                   =   relation_expression, {logic_operator,
↪  relation_expression};

relation_expression             =   additive_expression, {relation_operator,
↪  additive_expression};

additive_expression               =   multiplicative_expression,
↪  {additive_operator, multiplicative_expression};

multiplicative_expression   =   unary_expression, { multiplicative_operator,
↪  unary_expression};

unary_expression                   =   ["-"], value_expression;

value_expression                     =        int
                                 float
                                 string
                                 grouped_expression
                                 assignable, [ function_call_expression ]

function_call_expression    =   assignable, '(', [arguments], ')';

grouped_expression          = '(', expression ')'
```

3 Typy wbudowane

Język akceptowany przez stworzony interpreter jest wyposażony w trzy typy wbudowane

- *dynamic*.
- *string*
- *datetime*,
- *period*.

Pierwszy z nich jest dynamicznym typem przyjmującym wymagane typy podstawowe (int, float, bool) w zależności od okoliczności. Dwa ostatnie są typami obiektowymi - posiadają atrybuty których wartość można odczytywać (a w przypadku datetime również ustawiać).

3.1 Obsługa dat – *datetime*

3.1.1 Atrybuty

- DynamicValue *year*,
- DynamicValue *month*,
- DynamicValue *day*,
- DynamicValue *hour*,
- DynamicValue *minute*,
- DynamicValue *second*.

3.1.2 Metody

- **konstruktor** - przyjmuje od 1 do 6 argumentów ustawiających wartości wszystkich atrybutów. Jedynym wymaganym parametrem jest *year*, reszta w przypadku nie zdefiniowana ustawiana będzie na wartości domyślne. W wypadku wywołania zbłędnymi parametrami (tzn. np. dzień posiadający powyżej 23 godzin) wyrzucany jest wyjątek.
- **przeciążenie operatorów logicznych '–'** - obiekty tego typu mogą występować jako argumenty operacji odejmowania. Wynikiem tej operacji będzie obiekt typu *period*, którego szczegółowy opis znajduje się w kolejnym podrozdziale. Ponadto na zmienne tego typu mogą być ze sobą porównywane - *mniejsza* data to ta, która jest starsza.
- **występowanie jako argument metody print** - wyposażenie w mechanizm formatowania tekstu do wypisu. Obiekt typu *datetime* będzie wypisywać się w formacie *year-month-day hour:minute:second*, przy założeniu że każdy z atrybutów składa się z dwóch cyfr (za wyjątkiem atrybutu *year*).

3.2 Obsługa okresu – *period*

- DynamicValue readonly *Days* - zwraca pełną liczbę dni w danym okresie,
- DynamicValue readonly *Hours* - zwraca pełną liczbę godzin w danym okresie,
- DynamicValue readonly *Minutes* - zwraca pełną liczbę minut w danym okresie,
- DynamicValue readonly *Seconds* - zwraca pełną liczbę sekund w danym okresie,

3.2.1 Metody

- **konstruktor** - przyjmuje od 0 do 4 argumentów ustawiających wartości wszystkich atrybutów. Domyślnie wszystkie atrybuty ustawiane są na 0.
- **przeciążenie operatorów '+' i '–'** - dane typu *period* można zarówno dodawać, jak i odejmować. Wynikiem takich operacji jest obiekt typu *period*.

W dokumentacji wstępnej zakładane było powstanie jeszcze jednego typu - Int, jednak nie został on zaimplementowany. Ponadto metody wewnątrz typów obiektowych obsługiwane są w pełni przez ich parametry, nie istnieją zdefiniowane metody typ *GetDays()*

4 Przykłady skryptów

W poniższym rozdziale znajdują się dwa przykładowe programy akceptowalne przez interpreter języka *Basilisk*

4.1 Przykład 1

```
inauguracja = datetime(2021,6,11,18,0)
meczZeSzwecja = datetime(2021,6,23,20,30)
okres = meczZeSzwecja - inauguracja
print("Polska brała udział w tegorocznym EURO od " + inauguracja + " do " +
↪  meczZeSzwecja)
print("Trwalo to " + okres.Days + " dni")
```

4.2 Przykład 2

```
def fibonacci(a):
{
if(a == 0 or a == 1):
{
return 1
}
return fibonacci(a - 1) + fibonacci(a - 2)
}
i = 0
while(i<10):
{
print(fibonacci(i))
i = i + 1
}
```