Proste a'la Python z definiowaniem własnych klas, atrybutów i metod - Dokumentacja wstepna

Krzysztof Pożoga

Listopad 2021

1 Opis funkcjonalności

Implementowanym jezykiem jest uproszczona wersja Python z definiowaniem własnych klas, atrybutów i metod. Wciecia zostały zastapione znakami "{" i "}". Wbudowane typy na jakie pozwala jezyk to int oraz string (tylko jako stała tekstowa w wywołaniach funkcji print).

1.1 Deklaracje zmiennych

Deklaracja ma forme: *nazwa_zmiennej* = *wartość*.
Aby deklaracja była poprawna po nazwie zmiennej musi pojawić sie przypisanie wartości. Zakładamy, że zmienna niebedaca atrybutem klasy żyje do końca bloku, w którym została zadeklarowana.

```
d = 3
```

1.2 Klasy

1.2.1 Deklaracje klas

Deklaracja klasy ma forme:

```
class class_name
{
    deklaracje_atrybutow_klasy
    deklaracje_metod
}
```

Atrybuty klasy deklarujemy w taki sam sposób jak zmienne. Sa one statyczne dla danej klasy i ich żywotność nie jest powiazana z żadna instacja danej klasy. Dostep do atrybutu musi być poprzedzony nazwa klasy i znakiem ".".

Dysponujac istniejaca instancja klasy możemy deklarować także atrybuty instancji klasy, które od standardowej deklaracji zmiennej odróżniaja sie tym, że sa poprzedzone nazwa instancji klasy i znakiem ".".

1.2.2 Deklaracje metod

Deklaracja metody ma forme:

```
def method_name (parameters_list)
{
    kod_wykonywalny
}
```

Metody moga też zwracać pewna wartość za pomoca *return*.

Każda metoda może być wywołana statycznie poprzez

class_name.*method_name*(*parameters*) lub dla istniejacej instancji klasy: *class_instance_name*.*method_name*(*parameters*).

Należy wziać pod uwage, że w drugim przypadku obiekt bedacy instancja klasy automatycznie przekazywany jest jako pierwszy parametr wywołania metody.

Dodatkowo metoda o nazwie *__init__*(*parameters*) nazywana też konstruktorem może być wywołana poprzez

class_name(*parameters*).

W tym przypadku tworzona jest pusta instancja klasy *class_name* i konstruktor wywoływany jest jako metoda tejże instancji.

Metody zadeklarowane w przestrzeni globalnej, nazywane też funkcjami, nie sa zwiazane z żadna klasa, a co za tym idzie nie moga być wywołane jako metoda instancji klasy. Przy wywołaniu funkcji nie pojawia sie też *class_name*.
Ponadto przyjmujemy, że funkcja o nazwie *main* jest punktem wejściowym programu.

1.2.3 Przykładowa klasa

```
class bike
{
    seial_number = 1
    __init__(self, price)
    {
        self.price = price
        self.serial_number = bike.serial_number
        bike.serial_number +=1
    }
}
```

1.3 Petle

1.3.1 Petla for

Wykonuje sie dla kolejnych liczb z pewnego zakresu.

for $i = 1 : 10$

```
  {
    kod_wykonywalny
  }
```

1.3.2 Petla while

Wykonuje sie dopóki jest spełniany pewien warunek.
$while\ i < 10$
{
$kod\_wykonywalny$
}

1.4 Instrukcja warunkowa if

Wykona sie jeśli warunek jest spełniony.
$if\ \ i > 5$
{
$kod\_wykonywalny$
}

1.5 Operatory

1.5.1 Operatory unarne

- minus unarny "-"

1.5.2 Operatory arytmetyczne

- dodawanie "+"
- odejmowanie "-"
- mnożenie "*"
- dzielenie "/"
- dzielenie modulo "%"

1.5.3 Operatory logiczne

- koniunkcja "&&"
- alternatywa "||"

1.5.4 Operatory porównania

- równy "=="
- różny "!="
- wiekszy ">"
- wiekszy lub równy ">="
- mniejszy "<"
- mniejszy lub równy "<="

1.5.5 Operatory przypisania

- przypisanie "="
- "+="
- "-="
- "*="
- "/="
- "%="

1.6 Komentarze

Dopuszczalne sa komentarze jednolinijkowe zaczynajace sie od symbolu "#".

1.7 Instrukcja wyjścia print

Print jest wbudowana funkcja postaci *print*(*params*), gdzie params jest lista argumentów bedacych stringami badź wartościami "assignable".

Assignable oznacza wszystko, co może być przypisane do zmiennej, a string oznacza ciag znaków zaczynajacy i kończacy sie znakiem ".

Dopuszczalne znaki specjalne:

- koniec lini "\n"
- tabulator "\t"
- backslash "\\"
- cudzysłów "\""

2 Gramatyka jezyka

```
digit = "0" – "9"
letter = "a"–"z" | "A"–"Z" | "_"
id = letter { digit | letter}
comment = "#" { ? any_char ? }
string = """ { ? any_char ? } """
number = digit { digit }
minus = "–"
assignOp = "=" | "+=" | "–=" | "*=" | "/=" | "%="
logicOp = "||" | "&&"
relOp = "==" | "!=" | "<" | ">" | "<=" | ">="
addOp = "+" \| "–"
```

```
multOp = "*" | "/" | "%"


program = { classDef | funDef }
classDef = "class" className "{" classBody "}"
className = id
classBody = { attributeDef | funDef }
attributeDef = varDef
varDef = fieldId "=" assignable
methodDef = "def" id "(" [ Ids ] ")" "{" executable [ret] "}"
Ids = id { "," id }
ret = "return" assignable
assignable = expression
fieldId = [ className "." ] { id "." } id
funCall = fieldId "(" [ arguments ] ")"
arguments = expression { "," expression }
expression = multiplicativeExpr { addOp multiplicativeExpr }
multiplicativeExpr = primaryExpr { multOp primaryExpr }
primaryExpr = [minus] ( number | fieldId | funCall ) | parenthExpr
parenthExpr = "(" expression ")"
assign = fieldId assignOp assignable
logicExpr = relExpr { logicOp relExpr }
relExpr = expression {relOp expression}
if = "if"  logicExpr "{" executable "}"
while = "while" logicExpr "{" executable "}"
for = "for" [id "="] assignable ":" assignable "{" executable "}"
executable =  { if | while | for | line }
line = varDef | assign | funCall
```

3 Obsługa błedów

Skaner i parser sa w stanie wskazać błedny leksykalnie fragment kodu. Błedy w czasie wykonania programu (dzielenie przez 0) powinny przerwać działanie programu ze zgłoszeniem wyjatku.

4 Testy

Testy jednostkowe dla każdego modułu i dla każdego tokenu

5 Uruchomienie programu

Projekt bedzie pisany w Visual studio 2019 na Win10 x64 jako aplikacja konsolowa. Wykorzystany zostanie jezyk c oraz wbudowany kompilator visuala.

Konsola przyjmuje input użytkownika lub plik tekstowy do kompilacji /interpretacji.

6 Realizacja

Skaner przetwarza wejście na tokeny. Parser analizuje tokeny i na ich podstawie buduje drzewo składniowe.

Interpreter przechodzi po drzewie i na jego podstawie układa kod wykonywalny.

Zaczynajac od korzenia powinniśmy rekursywnie przejść wszystkie elementy drzewa.

7 Prosty program

```
class samochod
{
    nastepny_numer_produkcyjny = 1337


    def __init__(self, price)
    {
        self.price = price
        self.numer_produkcyjny = samochod.nastepny_numer_produkcyjny
        samochod.nastepny_numer_produkcyjny +=1
    }

    def czyDrozszy(self, sam)
    {
        return self.price-sam.price
    }
}

sam = samochod(1000)
sam2 = samochod(2000)

if sam.czyDrozszy(sam2)
{
    sam2 = sam
}

print(sam2.numer_produkcyjny)
```