ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ ΣΤΙΣ ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ2020

ΜΕΛΗ ΟΜΑΔΑΣ

ΤΣΑΜΠΑΣ ΣΤΥΛΙΑΝΟΣ 1039884

ΣΙΑΜΟΓΛΟΥ ΧΑΡΑΛΑΜΠΟΣ 1041601

ΑΝΝΑ ΜΑΡΙΑ ΖΑΧΑΡΟΠΟΥΛΟΥ 1047139

ΤΣΑΒΑΛΑΣ ΒΑΣΙΛΗΣ 5907

Περιγραφή

Στόχος της άσκησης είναι η κατασκευή ενός parser για ένα υποσύνολο της γλώσσας python. Για αυτό τον λόγο ακολουθήσαμε τις παρακάτω συμβάσεις.

Τα σχόλια θεωρούμε ότι ξεκινούν με το σύμβολο ‘#’ μέχρι το τέλος της γραμμής όπως παρακάτω:

```
# ... σχολιο
```

Ως αλφθαριθμητικά θεωρούμε τα παρακάτω:

```
“string1”
‘string2’
```

Ως αναθέσεις θεωρούμε εκφράσεις όπως:

```
a = x
a = (x + y) * z
a = ”test1” + ”TeSt2” + a
a = fun() + fun(a)
```

Για εκτύπωση χρησιμοποιείται η εντολή `print()`.

Για τις εντολες if, for χρησιμοποιούμε την παρακάτω σύνταξη:

```
if (condition):
      block_body
for i in <iter>:
      block_body
```

Όπου `<iter>` μπορεί να είναι μία κλήση σε συνάρτηση πχ. `range(10)`, `range(2, 20)`, ένας iterator `A`, μία λίστα `[3, 5, 7, 9]`.

Οι δηλώνονται με την παρακάτω σύνταξη:

```
def func(a, b):
      function_body
```

Οι κλάσεις δηλώνονται με την σύνταξη:

```
class ClassName:
      variables
      function_definitions
```

Η εισαγωγή `modules` μπορεί να γίνει με τους παρακάτω τρόπους:

```
import module
from module import function1, function2
import module as name
```

Υποστηρίζονται λίστες και dictionaries με την παρακάτω σύνταξη:

```
list1 = [1, “string”, ‘string’]
dict = {
      ‘key’:  value,
       key : ‘value’,
      ‘key’: ‘value’
}
```

Επίσης αναγνωρίζονται οι εντολές `items()` και `setdefault()` των dictionaries.

Γραμματική BNF

```
/** Complete file */
main_part:
    parts | main_part parts ;

/** Distinct program blocks */
parts:
    command | modules | class_def ;

/** Different commands */
command:
    function | print | assignment | if | for | return ;

id_parts:
    ID | ID '.' ID ;

/** Module imports */
modules:
    KEY_IMPORT module_list | KEY_FROM id_parts KEY_IMPORT var_lists ;
module_list:
    module_name | module_list ',' module_name ;
module_name:
    id_parts | id_parts KEY_AS ID ;

/** Class definitions */
class_part:
    command | func_def | class_part command | class_part func_def;
class:
    KEY_CLASS ID ':' class_part;
class_def:
    class | func_def;

/** Function definitions */
func_def:
    KEY_DEF function ':' command ;

/** Expression list used as input variables in functions */
var_lists:
    expr_vals | var_lists ',' expr_vals;

/** Variable assignment */
assignment:
    id_parts '=' expr_vals | id_parts '=' lambda;

/** Print specific parser */
print:
    KEY_PRINT '(' multi_expr_vals ')' ;

/** Return keyword specific parser */
return:
    KEY_RETURN multi_expr_vals | KEY_RETURN ;

/** If keyword parser */
if:
    KEY_IF cond_vals ':' command;

/** For keyword parser */
for:
    KEY_FOR ID KEY_IN KEY_RANGE '(' expr_vals ')' ':' command
    | KEY_FOR ID KEY_IN ID ':' command
    | KEY_FOR ID KEY_IN ID ':' list ;

/** Lambda expression parser */
```

```
lambda:
    KEY_LAMBDA var_lists ':' expr_vals;

/** setdefault and items specific parser */
setdefault:
    ID '.' KEY_SETDEFAULT '(' type ')'
    | ID '.' KEY_SETDEFAULT '(' type ',' type ')' ;

items:
    ID '.' KEY_ITEMS '(' ')' ;

/** List of possible expressions in python */
expr_vals:
    expr_parts
    | list
    | dict
    | setdefault
    | items
    | expr_vals operator expr_parts
    | expr_vals operator list
    | expr_vals operator dict
    | '(' expr_vals ')' ;

/** List parser */
list:
    '[' multi_expr_vals ']';

/** Dictionary parser */
dict:
    '{' dict_parts '}' ;
dict_parts:
    dict_kvpair | dict_parts ',' dict_kvpair ;
dict_kvpair:
    expr_parts ':' expr_parts ;

/** Comma-separated expressions and function prototype*/
multi_expr_vals:
    expr_parts | multi_expr_vals ',' expr_parts ;
expr_parts:
    id_parts | function | type;
function:
    id_parts '(' ')' | id_parts '(' var_lists ')'

/** Variable types */
type:
    STR | CHARS | INTEGER | DOUBLE ;

/** Conditional expressions */
cond_vals:
    expr_vals conditional expr_vals
    | '(' cond_vals ')'
    | cond_vals KEY_AND expr_vals
    | cond_vals KEY_OR expr_vals;

/** Arithmetic operators */
operator:
    '+' | '-' | '*' | '/' | '%' ;

/** Conditional operators */
conditional:
    '>' | '<' | NOTEQUAL | EQUAL | GT | LT ;
```

Λεκτικός αναλυτής

```
    /* symbols */
[ \t] ;
"#"(.)*[\n] {NLines++;};

"+"|"-"|"*"|"/"|":"|";"|"["|"]"|"("|")"|"{"|"}"|"="|","|"." { return
yytext[0]; }

">="        return GT;
"<="        return LT;
"=="        return EQUAL;
"!="        return NOTEQUAL;
"\""        return DQT;
"\'"        return SQT;
[\n]        {NLines++; };

    /* keywords */
"import"        return KEY_IMPORT;
"from"          return KEY_FROM;
"as"            return KEY_AS;
"class"         return KEY_CLASS;
"def"           return KEY_DEF;
"return"        return KEY_RETURN;
"print"         return KEY_PRINT;
"if"            return KEY_IF;
"and"           return KEY_AND;
"or"            return KEY_OR;
"for"           return KEY_FOR;
"in"            return KEY_IN;
"range"         return KEY_RANGE;
"items"         return KEY_ITEMS;
"setdefault"    return KEY_SETDEFAULT;
"lambda"        return KEY_LAMBDA;

    /* data types */
[a-zA-Z_][a-zA-Z0-9_]*          { return ID; }
[-|+]*[0-9][0-9]*               { return INTEGER; }
[-|+]*[0-9][0-9]*"."[0-9][0-9]* { return DOUBLE; }
[0-9a-zA-Z_]*                   { return CHARS; }
\"[^"\n]*\"|\'[^'\n]*\'         { return STR; }
```

Δοκιμαστικό πρόγραμμα

```
import os
import sys, pandas as pd, pprint
import pandas.api
from numpy import NaN

list0 = [1, 'test']
key0 = tuple(list0)
value0 = [2, "test"]
key3 = int()
dict0 = { key0: value0}
dict0 = {"key1": "value1"}
dict0 = {
    "key1": "value1",
    "key2": "value2",
    key3: 19.64
}

z = dict0.items()
z = dict0.setdefault("key1", "value0")

key4 = tuple([1, 2])

# comment
# second comment
class class0:
    a1 = 1

    def seta(self, a1):
        a = a1 * 2
        print(a)
        return a


def func0():
    return


def func1(x):
    y = 2 * x
    z = 5 * y
    return z


def func2(x):
    l = lambda x: x + 1
    x = l(x)
    for i in range(10):
        i = i + 1
    func1(x)
    func0()
    return x


x = 1
cls0 = class0()
cls0.a1 = 3
print("func1:", func1(x))
print("func2:", func2(x))
print("class:", cls0.seta(x))
```

Το δοκιμαστικό πρόγραμμα αναγνωρίζεται ως σωστό από τον parser μας και εκτελείτε χωρίς λάθη από τον διερμηνέα της python.

Παράδειγμα εκτέλεσης