الجلسة الثانية: الماسح

الهدف من الجلسة

التعريف بمكتبة PLY وأداة المسح lex.

بناء برنامج يحاكي عمل الماسح.

مكتبة ply

تعد هذه المكتبة تطبيق فعلي لأداتي lex أوyacc بلغة البايثون مع الحفاظ على خصائص عمل الأداتين الأساسي مثل الإعراب بخوارزمية LALR(1) واكتشاف الأخطاء.

تتكون من وحدتين أساسيتين هما lex.py وyacc.py اللواتي يكملن عمل بعضهما.

الأداة lex تؤمن واجهة لتوليد الـ tokens.

الأداة yacc تستخدم هذه الـ tokens لإنشاء القواعد اللغوية.

أداة lex

تقوم بمسح سلسلة الدخل وتقسيمها إلى رموز tokens (وهي أصغر وحدة ذات معنى في النص).

مثلاً إذا كان الدخل على الشكل التالي:

```
x = 3 + 42 * (s - t)
```

سيقوم الماسح بتوليد الـ tokens التالية:

```
'x','=', '3', '+', '42', '*', '(', 's', '-', 't', ')'
```

تُعطى الرموز أسماءً تعبر عنها، مثلاً:

```
'ID','EQUALS','NUMBER','PLUS','NUMBER','TIMES',
'LPAREN','ID','MINUS','ID','RPAREN'
```

الشكل الفعلي سيكون كأزواج:

```
('ID','x'), ('EQUALS','='), ('NUMBER','3'),
('PLUS','+'), ('NUMBER','42), ('TIMES','*'),
('LPAREN','('), ('ID','s'), ('MINUS','-'),
('ID','t'), ('RPAREN',')'
```

يتعرف الماسح على الرموز من خلال التعابير النظامية، وفي لغة البايثون نقوم باستخدام الوحدة re لتشكيل التعبير النظامي.

مثال:

```
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t
```

يأخذ هذا التابع بارمتروحيد يعبر عن الرمز t وهو من نوع LexToken.

يحددهذا نوع الرمز t من خلال السطر البرمجي:

r'\d+'

وقيمة الرمز من خلال السطر البرمجي:

t.value=int(t.value)

ملاحظات:

- في حال تطلب البرنامج التعبير عن الفراغ كتعبيرنظامي، يستخدم S\.
- للتعبيرعن # يمكن استخدام التعبير النظامي #\.
- الرموز التي يُعبر عنها بتوابع تضاف بنفس الترتيب الذي تظهر عليه في ملف الماسح.
- الرموز التي يُعبر عنها كسلاسل نصية تُضاف بترتيب تنازلي حسب طول السلسلة مما يضمن التعرف الصحيح على الرموز، مثال عنها الفرق بين '=' والـ '==' يجب فحص '==' أولاً.

التوابع الأساسية لعمل الـlexer:

اسم التابع	عمل التابع
lexer.lineno	رقم السطر الحالي
lexer.lexpos	الموقع الحالي في سلسلة الدخل
lexer.lexdata	قيمة النص الحالي من بيانات الدخل
Lexer.input	إدخال النص إلى الماسح

بعض التعابير النظامية الشائعة:

العبارة	التطابقات
abc	abc
abc*	ab,abc,abcc,abccc,...
abc+	abc,abcc,abccc,...
a(bc)+	abc,abcbc,abcbcbc,...
a(bc)?	a,abc
[abc]	a,b,c
[a-z]	أي حرف بين a وz
[a\-z]	a,-,z
[-az]	-,a,z
[A-Za-z0-9]+	حرف أو أكثر(بما في ذلك الأعداد)
[\t\n]+	الفراغات
[^ab]	أي شيء باستثناء a و b
[a^b]	a,^,b
[a|b]	a,|,b
a|b	a أو b

مكونات ملف الماسح لمسح كود يعبر عن آلة حاسبة:

1. الرموز التي نحتاجها للتعرف على كل أجزاء البرنامج.

```
tokens = ('NUMBER','PLUS', 'MINUS', 'TIMES',
          'DIVIDE','LPAREN', 'RPAREN','GREATERE','SEMI','EQUAL',
          'LPRACT','RPRACT','ID','KEYWORD')

# Reserved keywords
keywords = {
    'if': 'KEYWORD',
    'then': 'KEYWORD',
    'begin': 'KEYWORD',
    'end': 'KEYWORD',
    'print':'KEYWORD',
    'int':'KEYWORD'}
```

رسم توضيحي 1 الرموز الموجودة في البرنامج

2. التعابير النظامية التي تعبرعن الاستجابة لقراءة token ما.

<u>للتعابير البسيطة:</u>

```
# Regular expression rules for simple tokens
t_PLUS    = r'\+'
t_MINUS   = r'\-'
t_TIMES   = r'\*'
t_DIVIDE  = r'\/'
t_LPAREN  = r'\('
t_RPAREN  = r'\)'
t_SEMI    =r'\;'
t_LPRACT=r'\{'
t_RPRACT=r'\}'
t_GREATERE=r'\>\='
t_EQUAL=r'\='
```

رسم توضيحي 2 الاستجابة للتعابير االبسيطة

<u>للتعابير الأكثر تعقيداً:</u>

```
# A regular expression rule for numbers
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)  # Convert to integer
    return t
# A regular expression rule for identifiers and keywords
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = keywords.get(t.value, 'ID')  # Check for reserved words
    return t

# Define a rule to track line numbers (optional)
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)
# A string containing ignored characters (spaces and tabs)
t_ignore  = ' \t'
# A rule to handle comments (single-line between { })
def t_comment(t):
    r'\{[^}]*\}'
    pass  # No return value -> token discarded
```

رسم توضيحي 3 الاستجابة للتعابير الأكثر تعقيداً

مثال كود برمجي متكامل

```python
import ply.lex as lex

# List of token names
tokens = ('NUMBER','PLUS', 'MINUS', 'TIMES', 'DIVIDE','LPAREN', 'RPAREN','ID','KEYWORD')
# Reserved keywords
keywords = {
    'if': 'KEYWORD',
    'then': 'KEYWORD',
    'begin': 'KEYWORD',
    'end': 'KEYWORD'}
# Regular expression rules for simple tokens
t_PLUS    = r'\+'
t_MINUS   = r'\-'
t_TIMES   = r'\*'
t_DIVIDE  = r'\/'
t_LPAREN  = r'\('
t_RPAREN  = r'\)'
# A regular expression rule for numbers
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)  # Convert to integer
    return t
# A regular expression rule for identifiers and keywords
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = keywords.get(t.value, 'ID')  # Check for reserved words
    return t
# A regular expression rule for identifiers and keywords
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = keywords.get(t.value, 'ID')  # Check for reserved words
    return t
# Define a rule to track line numbers (optional)
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)
# A string containing ignored characters (spaces and tabs)
t_ignore  = ' \t'
# A rule to handle comments (single-line between { })
def t_comment(t):
    r'\{[^}]*\}'
    pass  # No return value -> token discarded
# Build the lexer
lexer = lex.lex()
# Test code
if __name__ == "__main__":
    data = "5 * 2+1"
    lexer.input(data)
    for tok in lexer:
        print(tok)
```

رسم توضيحي 4 ملف الماسح للتعبير عن آلة حاسبة بسيطة

تمارين:

- عدل الكود السابق ليتعرف الماسح على عملية باقي القسمة الـ mod
- عدل الكود السابق ليتعرف الماسح على الكود التالي:

```
i(x>5) print("hello");
```

انتهت الجلسة

إعداد:م. ريم جبيلي