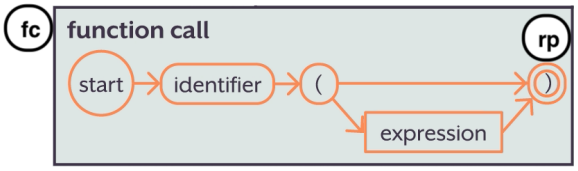
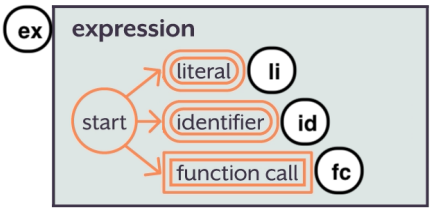
Syntax Diagrams and Sample Problem (expression statement, expression and function call)

In the video for lesson 2.04, syntax analysis started with the expression syntax diagram and a function call diagram was also used. This was done to simplify the explanation. However, syntax analysis actually begins with a statement and so far, the only statement that you have seen is an expression statement. The syntax diagram for expression statement is shown here along with the expression and function call syntax diagrams.

This reading contains a sample problem that applies syntax analysis to an expression statement using these syntax diagrams.

Current syntax diagrams:





Sample syntax analysis problem:

You will be asked to perform syntax analysis on a syntactically valid Python expression statement, using the current syntax diagrams. Each diagram is labeled with abbreviations you must use in your answers. The abbreviation on the left of a diagram denotes the diagram itself. For example, **ex** denotes the expression diagram. The abbreviations inside the diagram denote

accepting states for that diagram. For example, **id** denotes the identifier accepting state in the expression diagram.

If a syntax diagram is started, a left angle bracket indicates that it has started and its two letter abbreviation indicates which rule. For example, starting the expression diagram is denoted by:

<ex

When a syntax diagram is completed, this is indicated by its two letter abbreviation, followed by a right angle bracket. However, every syntax diagram completes at an accepting state so the **@** character and the two letter abbreviation for the accepting state are appended. For example, completing the expression diagram at the identifier accepting state is denoted by:

ex>@id

Here is a sample syntax analysis problem and its solution. The problem consists of 8 questions.

Question 1: This Python statement has valid syntax:

```
print('hello')
```

What is the first syntax diagram that is started?

We use annotations to help solve this problem. The annotations are removed to create the final answer. You should also use annotations to help you obtain your answers and then remove them when you enter your answers in a quiz.

The interpret always starts with a statement so the interpreter tries to match **print('hello')** to an expression statement. This is denoted:

```
<es trying to match print('hello')
```

We add what the interpreter is trying to match as an annotation at the end of this line, to help keep track of what is going on. The answer does not include the annotation: trying to match print('hello')

The answer is:

<es

Question 2: This Python statement has valid syntax:

```
print('hello')
```

What is the next syntax diagram that is started or completed?

Every expression statement diagram contains a single expression non-terminal state, so the expression diagram is started next. This is denoted:

```
<es trying to match print('hello')

  <ex trying to match print('hello')
```

The indentation indicates that the expression diagram is used from inside the expression statement diagram. No individual tokens have been matched to terminal states yet so the interpreter is still trying to match the entire statement.

The answer does not include the annotation: trying to match print('hello')

The answer is:

<ex

Question 3: This Python statement has valid syntax:

```
print('hello')
```

What is the next syntax diagram that is started or completed?

The expression diagram supports several different kinds of expressions, each represented by its own state. Since **print('hello')** is a function call, it should match the function call state. Since this is a non-terminal state, the function call diagram is used next. This is denoted:

```
<es trying to match print('hello')
```

```
  <ex trying to match print('hello')
```

```
    <fc trying to match print('hello')
```

The answer does not include the annotation: trying to match print('hello')

The answer is:

<fc

Question 4: This Python statement has valid syntax:

```
print('hello')
```

What is the next syntax diagram that is started or completed?

In the function call diagram, the next tokens are **print** and **(**, so they match the identifier and the left paren non-accepting terminal states. Since these tokens have been matched, they are removed from the tokens to match. However, non-accepting terminal states are not included in quiz answers, since only the start and completion of syntax diagrams are recorded. The next token to match is **'hello'** and the next state in the function call syntax diagram is the expression non-terminal state, which is expanded using the expression syntax diagram. This is denoted:

```
<es trying to match print('hello')
```

```
  <ex trying to match print('hello')
```

```
    <fc trying to match print('hello')
```

```
      <ex trying to match 'hello'
```

The answer does not include the annotation: trying to match 'hello'

The answer is:

<ex

Question 5: This Python statement has valid syntax:

```
print('hello')
```

What is the next syntax diagram that is started or completed?

The expression diagram supports several different kinds of expressions, each represented by its own state. The expression diagram contains terminal states such as literal, and non-terminal states, such as function call. All of the states are accepting states. When an accepting state in a syntax diagram is matched, the abbreviated name of the diagram, a right angle bracket, an **@** character and the two letter abbreviation of the accepting state are used.

In this example, the literal string **'hello'** matches the literal accepting state to complete the expression diagram. This is denoted:

```
<es trying to match print('hello')
  <ex trying to match print('hello')
    <fc trying to match print('hello')
      <ex trying to match 'hello'
      ex>@li matches 'hello'
```

The answer does not include the annotation: matches 'hello'

The answer is:

ex>@li

Question 6: This Python statement has valid syntax:

```
print('hello')
```

What is the next syntax diagram that is started or completed?

Since the expression diagram is complete, the interpreter returns to the function call diagram. The next token is **)** so it matches the right paren state. The right paren is an accepting state in the function call diagram. This is denoted:

```
<es trying to match print('hello')
```

```
  <ex trying to match print('hello')
    <fc trying to match print('hello')
      <ex trying to match 'hello'
      ex>@li matches 'hello'
    fc>@rp matches print('hello')
```

The answer does not include the annotation: matches print('hello')

The answer is:

fc>@rp

Question 7: This Python statement has valid syntax:

```
print('hello')
```

What is the next syntax diagram that is started or completed?

Since the function call diagram is complete, the interpreter returns to the expression diagram. Since the function call state is an accepting state, the expression diagram is also complete. This is denoted:

```
<es trying to match print('hello')
  <ex trying to match print('hello')
    <fc trying to match print('hello')
      <ex trying to match 'hello'
      ex>@li matches 'hello'
    fc>@rp matches print('hello')
  ex>@fc matches print('hello')
```

The answer does not include the annotation: matches print('hello')

The answer is:

ex>@fc

Question 8: This Python statement has valid syntax:

```
print('hello')
```

What is the next syntax diagram that is started or completed?

Since the expression diagram is complete, the interpreter returns to the expression statement diagram. Since the expression state is an accepting state, the expression statement diagram is also complete. This is denoted:

```
<es trying to match print('hello')
  <ex trying to match print('hello')
    <fc trying to match print('hello')
      <ex trying to match 'hello'
      ex>@li matches 'hello'
    fc>@rp matches print('hello')
  ex>@fc matches print('hello')
es>@ex matches print('hello')
```

The answer does not include the annotation: matches print('hello')

The answer is:

es>@ex

Note that there are four left angle brackets, one for each time a syntax diagram starts, and four right angle brackets, one for each time a syntax diagram successfully completes. The number of

left angle brackets must equal the number of right angle brackets, since each syntax diagram that is used must start and then complete at an accepting state.