Lecture 3: Operators, Expressions, Types

CS 51P

January 27, 2020

State

- State is a fundamental programming concept
  - State means you can
    - Save the meaning of something by giving it a name
    - Use (retrieve) the meaning of something by naming it
    - Combine these two actions ("operations") to build more complicated meaning

```
X = 2  # Read as "Assign 2 to X"
Y = 1  # Read as "Assign 1 to Y"
Z = X + Y  # ???
print(X,Y,Z)
```

visualize

State

- More State assignment (visualize)

```
message = 'And now for something completely different'

n = 17

pi = 3.14159
```

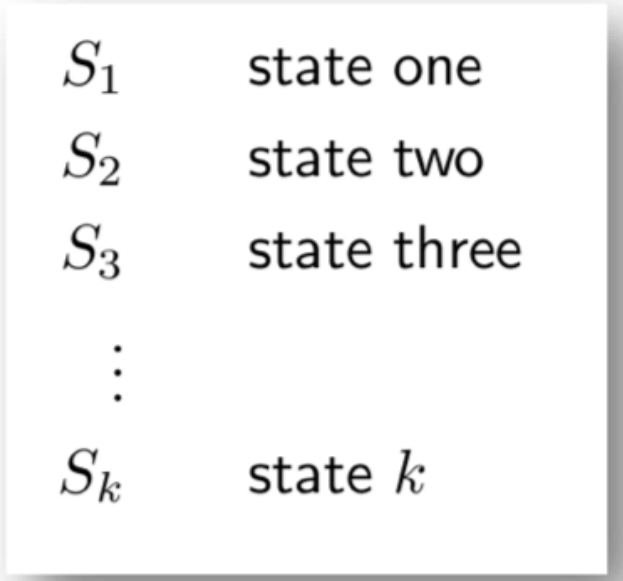
An Aside on Programming

- **Programming** is the manipulation of symbols
- **Syntax** defines the rules for combining symbols
- **Semantics** defines the "meaning" of a program

- A program can be syntactically wrong
- A syntactically correct program can be semantically wrong
- **Debugging** is a process for fixing syntax and semantic errors

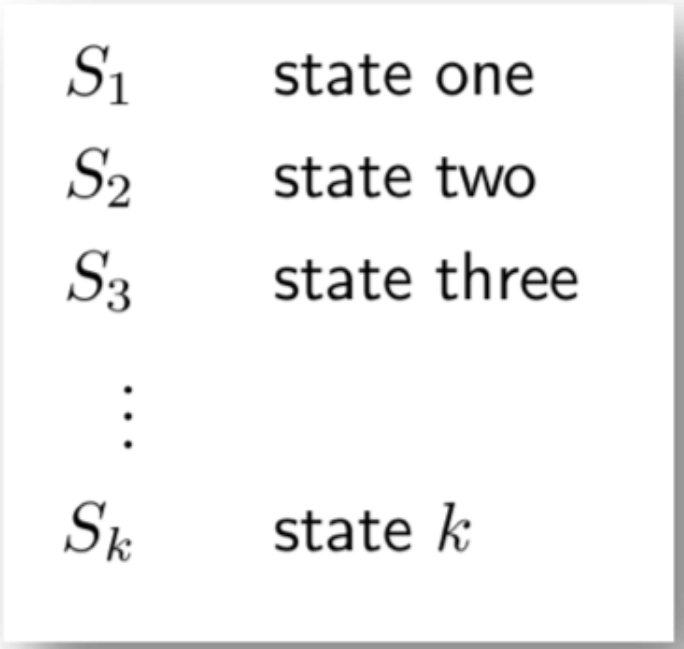
State

- State is a fundamental programming concept
- State is *memory* – a way to store information
- State is created/modified by a sequence of statements
  - This is a *serial* operation
  - There is an order
  - Retrieval at k depends upon an earlier store
- State requires a *name*



State

- Programs start from the topmost line
- First line is executed
- Then the second line is executed
- Then ...



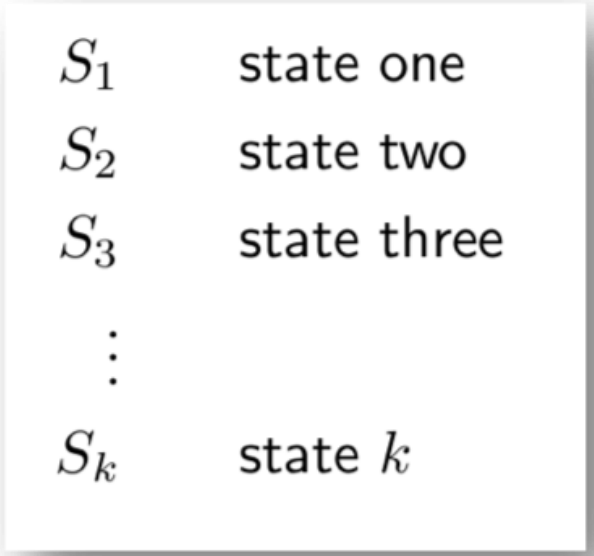
Example (Visualize)

```
x = 10    # S1
y = 5     # S2
z = x + y # S3
print(z)  # S4
```

State: Class Exercise (visualize)

- Label the states and either give the output or identify the bug
- If there's a bug, try to identify the bug and then fix it

```
x = 1
w = 3 + y + x
print(w)
y = 2
```



State: Class Exercise

```
x = 1
y = 2
w = 3 + y + x
print(w)

y = 2
x = 1
w = 3 + y + x
print(w)

x = 1
y = 2
z = y + x
w= z + 3
print(w)
```

- Treat this a three separate programs
- Show what the "state" consists of after each statement
- What is printed ?

Class Exercise

- Hal, Sue, Abe, and Lou have landed on a desert island are in search of coconuts
  - Hal finds 10
  - Sue finds twice as many as Hal
  - Abe finds one-tenth as many as Hal
  - Lou finds as many as Sue and Abe together
- Write code that finds the total number of coconuts.

```
Hal = 10

# Your code


print(total)
```

State*ment* (visualize)

- **Syntax** (Assignment)

variable = expression

- **Semantics**
  - Puts the value of *expression* in the location with variable *name*

```
x3 = 4
cat = 10 + x3
x3 = x3*2
print(x3)
```

Rules for "naming"

- Names can be as long as you like and can consist of letters, numbers and '_'
- *But* they can't start with a number
- Some names are *reserved*, e.g.
  - False, True
  - While, if
- Using illegal names will cause a "syntax error"

```
>>> 76trombones = 'big parade'
File "<stdin>", line 1
76trombones = 'big parade'
^
SyntaxError: invalid syntax
>>> more@ = 1000
File "<stdin>", line 1
more@ = 1000
^
SyntaxError: invalid syntax
```

Statements and Expressions

- A **statement** is a unit of code that has an effect

```
>>> n = 17 # assign 17 to 'n'
>>> print(n) # print the value of 'n'
```

- An **expression** is a combination of values, variables, and operators

```
>>> 42
42
>>> n
17
>>> n + 25
42
```

Debugging Redux

- There are three general categories of errors:
  - **Syntax error** : a violation of the structure of a program
  - **Runtime error**: an error that appears when the program executes
  - **Semantic error**: a problem relating to the meaning of a program
- Debugging is the process of finding and crushing these errors. Typically, it is necessary to resolve these errors in the order given above

Syntax errors

```
x = (1 + 3)
y = 8)
```

```
File "<ipython-input-1-3ccc20f90364>", line 2
y = 8)
^
SyntaxError: invalid syntax
```

Runtime Errors

```
x = 1
y = x + z
z = 2
```

```
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
NameError: name 'z' is not defined
```

Semantic Errors

- Semantic errors are typically the hardest to identify and find the *root cause*. Often, we use systematic testing to try to eliminate semantic errors

```
radius = 3
area_of_circle = radius**2
```

Types

- Numbers
  - Integers : -1, 400, 0, 24, …
  - Floats: 0.32, -1.141, 3.14159
- Strings
  - 'b', '-1', 'hello_kitty'
  - '"h"', '(1+2) + "dog"'
  - ""
  - "

```
>>> type(-1)
<class 'int'>

>>> type(0.32)
<class 'float'>

>>> type('')
<class 'str'>

>>> type('"h"')
<class 'str'>

>>> type(input("give it to me: "))
give it to me: 3.14159
<class 'str'>

type(int("33"))
<class 'int'>

>>> type(float("33"))
<class 'float'>
```

Integers, Floats, and Strings

- These are the three **types** we will use initially
- A **type** means:
  - Syntax – written structure
  - Semantics – meaning (how to interpret it)
  - Operations – what can be done with it

Integers

$$
\begin{array}{rcl}
\langle non\_zero\_integer \rangle & ::= & \mathbf{1}|\mathbf{2}|\mathbf{3}|\mathbf{4}|\mathbf{5}|\mathbf{6}|\mathbf{7}|\mathbf{8}|\mathbf{9} \\
\langle zero \rangle & ::= & \mathbf{0} \\
\langle digit \rangle & ::= & \langle zero \rangle | \langle non\_zero\_integer \rangle \\
\langle integer \rangle & ::= & \langle zero \rangle | \langle non\_zero\_integer \rangle \langle digit \rangle^{*} \\
\langle signed\_integer \rangle & ::= & [\mathbf{-}]^{+} \langle integer \rangle
\end{array}
$$

- Syntax
  - 0
  - All other integers must start with 1,2,..,9 followed by 0-9 any number of times
  - An integer may be prefixed with a '-'
- Semantics (what you learned in grade school)
- Operations
  - Addition
    - Syntax: x + y or (x + y)
    - Semantics: you got this one

Integers

```
print(0)
print(-1)
print(12200)
print(01)
```

$$
\begin{array}{rcl}
\langle non\_zero\_integer \rangle & ::= & \mathbf{1}|\mathbf{2}|\mathbf{3}|\mathbf{4}|\mathbf{5}|\mathbf{6}|\mathbf{7}|\mathbf{8}|\mathbf{9} \\
\langle zero \rangle & ::= & \mathbf{0} \\
\langle digit \rangle & ::= & \langle zero \rangle | \langle non\_zero\_integer \rangle \\
\langle integer \rangle & ::= & \langle zero \rangle | \langle non\_zero\_integer \rangle \langle digit \rangle^{*} \\
\langle signed\_integer \rangle & ::= & [\mathbf{-}]^{+} \langle integer \rangle
\end{array}
$$

```
File "<ipython-input-1-a7229a104295>", line 4
print(01)
^
SyntaxError: invalid token
```

```
print(123+)
```

```
File "<ipython-input-1-55b6482b84a9>", line 1
print(123+)
^
SyntaxError: invalid syntax
```

Strings

- Syntax
  - Must both start and end with either " or '
  - Empty string "" or ''
  - Inside quotes – anything from key board (quotes are a special case)
- Semantics： it's just text ！
- Operations:
  - Add: "ab" + "cd" creates the string "abcd" -- addition is concatenation
  - Multiply: integer * string or string * integer
    - 3* 'cat' creates 'catcatcat'
    - 'cat' * 3 creates 'catcatcat'

"Mi" + 2*"ssi" + "ppi"
"ba" + 2*"na"