LECTURE 5

LISTS AND TUPLES

MCS 260 Fall 2020

Emily Dumas

REMINDERS

- Quiz 2 available, Due 6pm central on Tue Sep 8
- Quiz 1 grades will be posted soon

MORE ON ASSIGNMENT

Recall that Python uses

```
name = value or expression
```

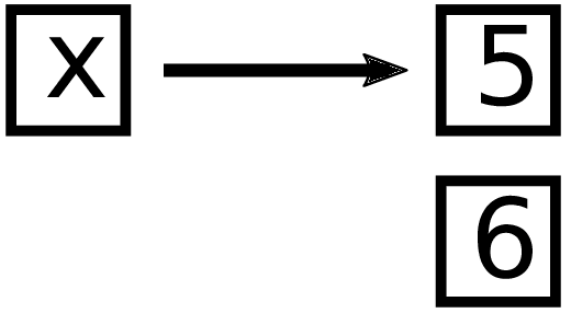
as syntax for assignments. The right hand side is evaluated first!

```
>>> x = 5
>>> x
5
>>> x = x + 1
>>> x
6
```

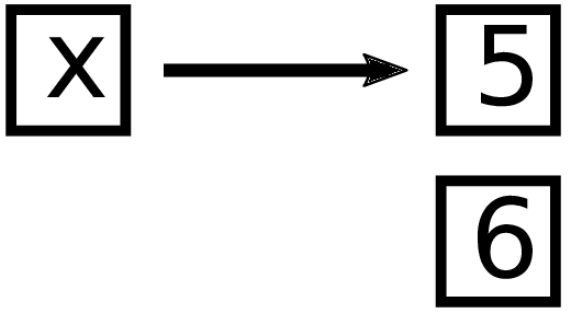

```
# previous line
x = x + 1
# next line
```


```
# previous line
x = x + 1
# next line
```

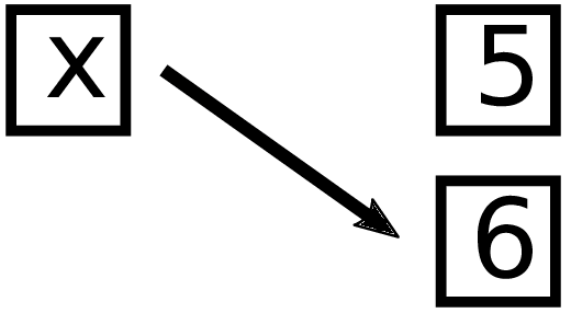

```
# previous line
x = x + 1
# next line
```



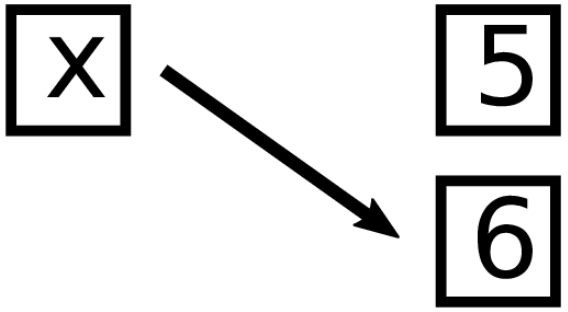

```
# previous line
x = x + 1
# next line
```



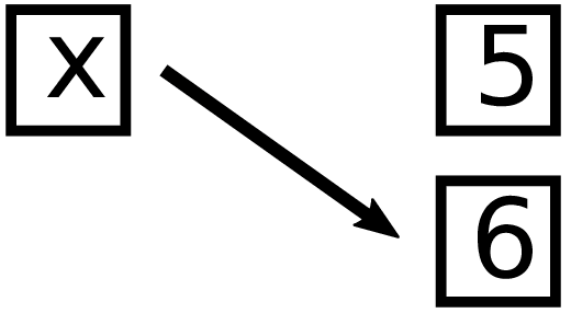

```
# previous line
x = x + 1
# next line
```



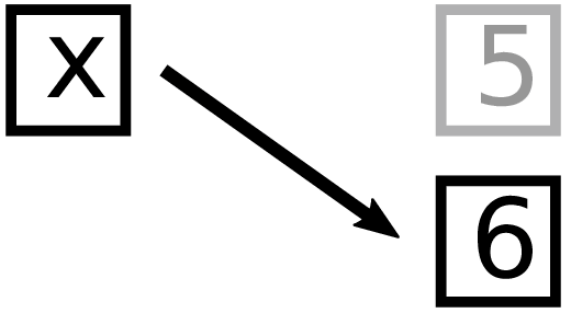

```
# previous line
x = x + 1
# next line
```



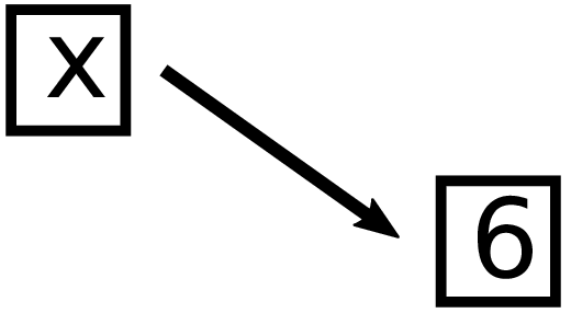

```
# previous line
x = x + 1
# next line
```




```
# previous line
x = x + 1
# next line
```




```
# previous line
x = x + 1
# next line
```




```
# previous line
x = x + 1
# next line
```

LISTS

A **list** is a sequence of values (of any types).

```
>>> L = [4, "red", 2.2, [5,6]]  # Square bracket = list
>>> L
[4, 'red', 2.2, [5, 6]]
>>> type(L)
<class 'list'>
>>> len(L)
4
```

Notice that the `len()` built-in supports lists.

The empty list exists and is written `[]`.

`+` and `*` operate similarly for lists as with strings.

```
>>> [1,2,3] + [4,5,6]
[1, 2, 3, 4, 5, 6]
>>> [1,99]*3
[1, 99, 1, 99, 1, 99]
```

Items can be retrieved by 0-based index:

```
>>> L = [4,8,15,16,23,42]
>>> L[2]
15
```

MUTABILITY

Lists are **mutable**, meaning that the contents can be changed.

```
>>> L = [4,8,15,16,23,42]
>>> L[2] = 999
>>> L
[4, 8, 999, 16, 23, 42]
```

An element of a list can be deleted with the **del** keyword. Note the indices of other elements change.

```
>>> L = [4,8,15,16,23,42]
>>> del L[2]
>>> L
[4, 8, 16, 23, 42]
```

You can't access or assign list indices that don't exist:

```
>>> L = [4,8,15,16,23,42]
>>> L[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> L[6] = 121
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

One way to add a new element to a list would be:

```
>>> L = L + [121]
>>> L
[4, 8, 15, 16, 23, 42, 121]
```

(Later we'll learn a faster way to do this.)

STRINGS ARE IMMUTABLE

In Python, strings are immutable. The characters can be accessed, but not changed.

```
>>> s = "it"
>>> s[1]
't'
>>> s[1] = "n"
Traceback (most recent call last):F
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

TUPLES

A **tuple** is a sequence of values (of any types). Like strings, tuples are immutable.

Tuples are entered as values separated by commas. Traditionally they are surrounded by parentheses, but this is not required. They support indexing and **`len`**().

```
>>> T = (2,6,"zero")  #  T = 2,6,"zero" also allowed
>>> T
(2, 6, 'zero')
>>> type(T)
<class 'tuple'>
>>> T[1]
6
>>> len(T)
3
```

A tuple with one element requires a trailing comma to distinguish it from an expression in grouping parentheses.

```
>>> T = (1)
>>> type(T)
<class 'int'>
>>> T = (1,)
>>> type(T)
<class 'tuple'>
```

The empty tuple exists and can be written as `()` or `tuple()`.

SEQUENCES

Strings, lists, and tuples are all examples of Python **sequences**: ordered collections of elements that can be retrieved by index.

```
>>> "asdf"[2]      # string indexing
'd'
>>> [1,2,3,4][2]   # list indexing
3
>>> (1,2,3,4)[2]   # tuple indexing
3
```

They all support `len()`.

NEGATIVE INDICES

Sequences allow negative indices, where -1 refers to the last element, -2 to the second to last, etc.

```
>>> "Oklahoma!"[-1]
'!'
>>> "Oklahoma!"[-3]
'm'
```

Can think of this as "wrap-around" behavior, with negative index meaning move to the left.

Negative indices (etc.) mean use of `len()` is rare.

SLICES

Sequences in Python support **slices** to retrieve (or assign) a segment.

The basic slice syntax is

```
x[i:j]
```

which retrieves elements of `x` with 0-based indices between `i` and `j`, *including* `i` *but not* `j`.

Either `i` or `j` can be omitted; missing `i` is taken to be 0, missing `j` is taken to be just past the end.

```
>>> s = "learning Python"
>>> s[:]
'learning Python'
>>> s[:-1]
'learning Pytho'
>>> s[1:4]
'ear'
>>> s[:5]
'learn'
>>> s[5:]
'ing Python'
```

l	e	a	r	n	i	n	g		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

SEQUENCE CONVERSION

The functions `list()` and `tuple()` convert other sequence types to list or tuple (resp.).

```
>>> list("abc")
['a', 'b', 'c']
>>> tuple( [1,2,3] )
(1, 2, 3)
>>> tuple("abc")
('a', 'b', 'c')
```

Careful: `str()` exists but doesn't convert a sequence to a string in the corresponding way.

SEQUENCE ASSIGNMENT

Python supports multiple variable assignments in one statement, with syntaxes:

```
    name0, name1, ..., nameN = SEQ
( name0, name1, ..., nameN ) = SEQ
[ name0, name1, ..., nameN ] = SEQ
```

The number of elements of sequence `SEQ` must match the number of names given. As usual, the right hand side is evaluated before assignment proceeds.

Swap two values:

```
>>> x=1
>>> y=8
>>> x,y = y,x
>>> x
8
>>> y
1
```

Not equivalent to two separate assignments:

```
>>> x=1
>>> y=8
>>> x=y
>>> y=x
>>> x
8
>>> y
8
```

Python is relatively unusual among programming languages for allowing this simple swap syntax.

In many other languages, an explicit temporary variable is needed, e.g.

```
temp = x
x = y
y = temp
```

SLICE ASSIGNMENT

A slice of a mutable sequence (list) can be assigned to another sequence, **even one of different length**.

The indicated slice is removed from the list and replaced with the elements of the sequence on the right hand side.

```
>>> L = [10,100,50,500]
>>> L[1:3]
[100, 50]
>>> L[1:3] = "math"
>>> L
[10, 'm', 'a', 't', 'h', 500]
```

REFERENCES

- In *Downey*:
  - Lists are covered in Chapter 10 (includes more material than in today's lecture)
  - Tuples are covered in Chapter 12 (includes more material than in today's lecture)
  - Section 12.2 discusses "tuple assignment", a particular example of the sequence assignment syntax we discussed today.

REVISION HISTORY

- 2020-09-02 Typos fixed
- 2020-09-01 Initial publication