Python basics II: lists, dictionaries and loops

Lists

A series of elements, initialized with []

```
list_1 = [] # empty list
list_2 = [1, 2] # a list of two
elements, first is 1 and second is 2
list_3 = [“element_1”, 1, [“sub”,
”list]] # lists can be composed on
any of the other types even other
lists!
```

List Operations

```
>>>[] # empty list
[]

>>>len([]) # get the number of elements
in a list
0

>>>a = [99, "hello", ["nested",
"list"]] # lists can store any value,
even other lists!
>>>print(a) # prints the values of the
list that is stored in variable 'a'
[99, 'hello', ['nested', 'list']]

>>>a = [0, 1]
>>>b = [2, 3]
>>>a + b # list addition, joins them
together
[0, 1, 2, 3]
```

```
>>>a * 3 # list multiplication creates
with duplicate elements
[0, 1, 0, 1, 0, 1]

>>>a = range(5) # creates a list with
elements 0 to 4
>>>print(a)
[0, 1, 2, 3, 4]

>>>a.append(5) # append adds a element
to end of a list
>>>print(a)
[0, 1, 2, 3, 4, 5]

>>>a.pop() # removes last element for
list and returns it
5
>>>print(a)
[0, 1, 2, 3, 4]
```

All operations: +, *, ==, [], .append(), .clear(), .copy(), .count(), .extend(), .index(), .insert(), .pop(), .remove(), .reverse()

Take 5 mins to try some operations, are there any surprises?

Dictionaries

A series of 'paired' elements, one is a unique key and one is a value. Instead of using indices like in lists to get values you can specify any value for a key. Dictionaries are defined using {} brackets

```
d = {'key1' : 'value1’}
d_1 = {0 : 1, 1: 2}
```

Dictionary Operations

```
>>>d = {'key1' : 'value1', 'key2' :
'value2'}
>>>print(d)
{'key2': 'value2', 'key1': 'value1'}

>>>len(d) # get number of items
2

>>>d['key1'] # get specific value by a
key
'value’

>>>d['key3'] # error for an invalid key
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'key3'
```

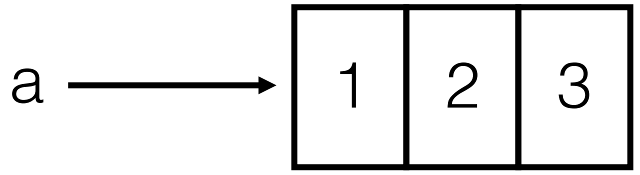
```
>>>d['key3'] = 'value3' # add new
key/value to dictionary
>>>d['key3']
'value3'

>>>d['key3'] = 'new_value' # override
previous value

>>>del d['key'] # removes key/value
pair from dictionary

>>>d.keys() # returns a list of keys in
the dictionary
['key2', 'key1']

>>>d.values() # returns a list of
values in the dictionary
['value2', 'value1']
```

All operations: `.append()`, `.clear()`, `.copy()`, `.pop()`, `.get()`, `.keys()`, `.values()`, `.items()`

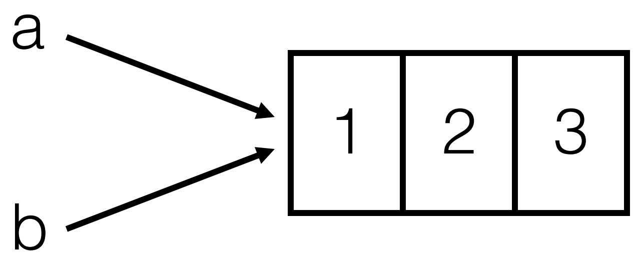
Take 5 mins to try some operations, are there any surprises?

Reference Semantics

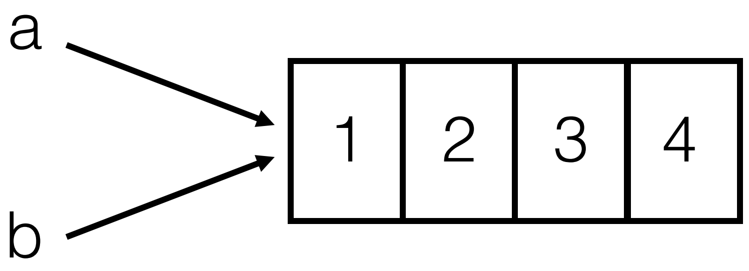
```
a = [1, 2, 3]
```



```
b = a
```



```
a.append(4)
```



Reference Semantics

```
>>>a = [1, 2, 3, 4]
>>>b = a  # now b is another reference or
'name' for the same list
>>>a.append(4)
>>>print(b)
>>>[1, 2, 3, 4]

# to create a new list instead of giving the
same list a new name
# any of these will work
>>> b = a.copy()
>>> b = list(a)
>>> b = a[::]
```

For loop

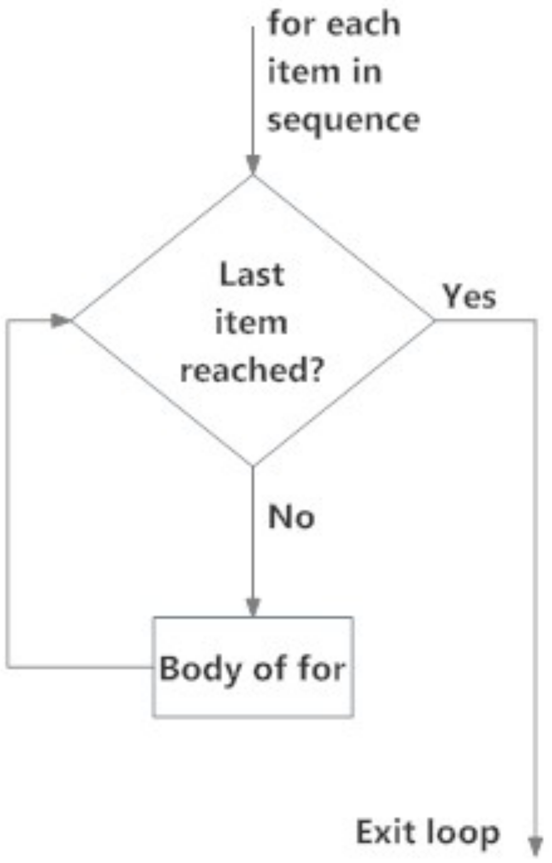


Fig: operation of for loop

```
# the format of a for loop
for element in list_of_elements:
   print(element) # would print each
element once

nums = [0, 5, 8, 9]
for n in nums:
   # checks to see if n is equal to 0
   if n == 0:
      print("n ="+n)

#output:
#n = 0
```

While loop

the while loop is a more generalized loop, it continue to repeat until its condition is NOT true

```
# will continue until count is equal to 5
count = 0
while count < 5:
    count += 1
    print(count)

#output
#1
#2
#3
#4
#5

while True:
    # infinite loop will never end
```

Break

break will exit from a loop once its reached

```
while True:
    print("looped")
    break # will only loop once
since exits here

#output
#looped

count = 0
while True:
    print(count)
    count += 1
    if count > 1:
        break

#output
#0
#1
#2
```

```
while True:
    while True:
        break # will only exit out
the of most recent loop, this will
still run forever
```

Continue

continue forces the next iteration of a loop

```
for i in range(5):
    continue
    print(i) # this is never 
reached, no output

for i in range(10):
    if i < 8:
        continue
    print(i)

#output
#8
#9
```

Loop else

an else at the end of a loop executes only when the loop has completely finished

```
for i in range(5):
    print(i)
else:
    print("finished loop")

#output
#0
#1
#2
#3
#4
#finished loop
```

```
for i in range(5):
    print(i)
    break # now the else will
not execute since this loop
did not finish
else:
    print("finished loop")

#output
#0
```