Python 3.2.3 (v3.2.3:3d0686d90f55, Apr 10 2012, 11:25:50)

[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin

Type "copyright", "credits" or "license()" for more information.

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

k in d[k]:

SyntaxError: invalid syntax

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k in d[k]:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 2, 2: 3, 3: 3})

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

count\_values\_that\_are\_keys({1: 2, 2: 3, 3: 3})

File "<pyshell#7>", line 4, in count\_values\_that\_are\_keys

if k in d[k]:

TypeError: argument of type 'int' is not iterable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k in d:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 2, 2: 3, 3: 3})

3

>>> count\_values\_that\_are\_keys({1: 2, 2: 3, 3: 0})

3

>>> count\_values\_that\_are\_keys({1: 2})

1

>>> count\_values\_that\_are\_keys({1: 1})

1

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if d in k:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#17>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#16>", line 4, in count\_values\_that\_are\_keys

if d in k:

TypeError: argument of type 'int' is not iterable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if d[k] in k:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#20>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#19>", line 4, in count\_values\_that\_are\_keys

if d[k] in k:

TypeError: argument of type 'int' is not iterable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k in k[k]:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#23>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#22>", line 4, in count\_values\_that\_are\_keys

if k in k[k]:

TypeError: 'int' object is not subscriptable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[k] in d[k]:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#26>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#25>", line 4, in count\_values\_that\_are\_keys

if k[k] in d[k]:

TypeError: 'int' object is not subscriptable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[k] in d:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#29>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#28>", line 4, in count\_values\_that\_are\_keys

if k[k] in d:

TypeError: 'int' object is not subscriptable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[i] in d:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#32>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#31>", line 4, in count\_values\_that\_are\_keys

if k[i] in d:

NameError: global name 'i' is not defined

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[i] in d[k]:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#35>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#34>", line 4, in count\_values\_that\_are\_keys

if k[i] in d[k]:

NameError: global name 'i' is not defined

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k in d[i]:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#38>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#37>", line 4, in count\_values\_that\_are\_keys

if k in d[i]:

NameError: global name 'i' is not defined

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k in d[k]:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#41>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#40>", line 4, in count\_values\_that\_are\_keys

if k in d[k]:

TypeError: argument of type 'int' is not iterable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[i] in k:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#44>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#43>", line 4, in count\_values\_that\_are\_keys

if k[i] in k:

NameError: global name 'i' is not defined

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[] in k:

result = result + 1

return result

SyntaxError: invalid syntax

>>>

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[ ] in k:

result = result + 1

return result

SyntaxError: invalid syntax

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if k[k] in k:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#50>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#49>", line 4, in count\_values\_that\_are\_keys

if k[k] in k:

TypeError: 'int' object is not subscriptable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if d[k] in k:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 1})

Traceback (most recent call last):

File "<pyshell#53>", line 1, in <module>

count\_values\_that\_are\_keys({1: 1})

File "<pyshell#52>", line 4, in count\_values\_that\_are\_keys

if d[k] in k:

TypeError: argument of type 'int' is not iterable

>>> def count\_values\_that\_are\_keys(d):

result = 0

for k in d:

if d[k] in d:

result = result + 1

return result

>>> count\_values\_that\_are\_keys({1: 2, 2: 3, 3: 0})

2

>>> count\_values\_that\_are\_keys({1: 1})

1

>>> count\_values\_that\_are\_keys({1: 2, 2: 3, 3: 3})

3

>>> def double\_last\_value(L):

s

SyntaxError: expected an indented block

>>>

def get\_negative\_nonnegative\_lists(L):

nonneg = []

>>> def get\_negative\_nonnegative\_lists(L):

nonneg = []

neg = []

for row in range(len(L)):

for col in range(len(L)):

if L[row][col] > 0:

nonneg.append(L[row][col])

else:

neg.append(L[row][col])

return (neg, nonneg)

>>> get\_negative\_nonnegative\_lists([[-1, 3, 5], [2, -4, 5], [4, 0, 8]])

([-1, -4, 0], [3, 5, 2, 5, 4, 8])

>>> def get\_negative\_nonnegative\_lists(L):

nonneg = []

neg = []

for row in range(len(L)):

for col in range(len(L)):

if L[row][col] < 0:

neg.append(L[row][col])

elif L[row][col] >= 0:

nonneg.append(L[row][col])

return (neg, nonneg)

>>> get\_negative\_nonnegative\_lists([[-1, 3, 5], [2, -4, 5], [4, 0, 8]])

([-1, -4], [3, 5, 2, 5, 4, 0, 8])

>>> def get\_negative\_nonnegative\_lists(L):

nonneg = []

neg = []

for row in range(len(L)):

for col in range(len(L)):

if 0 <= L[row][col]:

nonneg.append(L[row][col])

else:

neg.append(L[row][col])

return (neg, nonneg)

>>> get\_negative\_nonnegative\_lists([[-1, 3, 5], [2, -4, 5], [4, 0, 8]])

([-1, -4], [3, 5, 2, 5, 4, 0, 8])

>>> def get\_negative\_nonnegative\_lists(L):

nonneg = []

neg = []

for row in range(len(L)):

for col in range(len(L)):

val = L[row][col]

if val<0:

neg.append(val)

else:

nonneg.append(val)

return (neg, nonneg)

>>> get\_negative\_nonnegative\_lists([[-1, 3, 5], [2, -4, 5], [4, 0, 8]])

([-1, -4], [3, 5, 2, 5, 4, 0, 8])

>>> def get\_negative\_nonnegative\_lists(L):

nonneg = []

neg = []

for row in range(len(L)):

for col in range(len(L)):

if L[row][col] < 0:

neg.append(L[row][col])

nonneg.append(L[row][col])

return (neg, nonneg)

>>> get\_negative\_nonnegative\_lists([[-1, 3, 5], [2, -4, 5], [4, 0, 8]])

([-1, -4], [-1, 3, 5, 2, -4, 5, 4, 0, 8])

>>> def get\_keys(L, d):

result = []

for k in L:

if k in d:

result.append(k)

return result

>>> get\_keys([1, 2, 'a'], {'a': 3, 1: 2, 4: 'w'})

[1, 'a']

>>> def reverse(s):

result = ''

i = len(s) - 1

while i >= 0:

result = result + s[i]

i=i+1

return result

>>>

>>> reverse('abc')

Traceback (most recent call last):

File "<pyshell#108>", line 1, in <module>

reverse('abc')

File "<pyshell#106>", line 5, in reverse

result = result + s[i]

IndexError: string index out of range

>>> def reverse(s):

result = ''

i = len(s) - 1

while i >= 0:

result = result + s[i]

i=s[i]+1

return result

>>> reverse('abc')

Traceback (most recent call last):

File "<pyshell#111>", line 1, in <module>

reverse('abc')

File "<pyshell#110>", line 6, in reverse

i=s[i]+1

TypeError: Can't convert 'int' object to str implicitly

>>> def reverse(s):

result = ''

i = len(s) - 1

while i >= 0:

result = result + s[i]

i=i-1

return result

>>> reverse('abc')

'cba'

>>> reverse('abc')

'cba'

>>> reverse('1234!')

'!4321'

>>> def both\_start\_with(s1, s2, prefix):

return s1.startswith(prefix) == s2.startswith(prefix)

>>> both\_start\_with(sdfg,ser,sd)

Traceback (most recent call last):

File "<pyshell#121>", line 1, in <module>

both\_start\_with(sdfg,ser,sd)

NameError: name 'sdfg' is not defined

>>> def both\_start\_with(s1, s2, prefix):

if s1.startswith(prefix) and s2.startswith(prefix):

return True

else:

return False

>>> both\_start\_with(sdfg,ser,sd)

Traceback (most recent call last):

File "<pyshell#128>", line 1, in <module>

both\_start\_with(sdfg,ser,sd)

NameError: name 'sdfg' is not defined

>>> both\_start\_with(sdfg,ser,s)

Traceback (most recent call last):

File "<pyshell#129>", line 1, in <module>

both\_start\_with(sdfg,ser,s)

NameError: name 'sdfg' is not defined

>>> both\_start\_with("sdfg","ser","s")

True

>>> def both\_start\_with(s1, s2, prefix):

return s1.startswith(prefix) == s2.startswith(prefix)

>>> both\_start\_with("sdfg","ser","s")

True

>>> both\_start\_with("sdfg","ser","t")

True

>>> def both\_start\_with(s1, s2, prefix):

if s1.startswith(prefix) and s2.startswith(prefix):

return True

else:

return False

>>> both\_start\_with("sdfg","ser","t")

False

>>> def both\_start\_with(s1, s2, prefix):

return s1.startswith(prefix) == s2.startswith(prefix) == prefix.startswith

>>> both\_start\_with("sdfg","ser","t")

False

>>> both\_start\_with("sdfg","ser","s")

False

>>> def both\_start\_with(s1, s2, prefix):

return s1.startswith == s2.startswith == prefix.startswith

>>> both\_start\_with("sdfg","ser","s")

False

>>> def both\_start\_with(s1, s2, prefix):

return s1.startswith(prefix) == prefix.startswith and s2.startswith(prefix) == prefix.startswith

>>> both\_start\_with("sdfg","ser","s")

False

>>> def both\_start\_with(s1, s2, prefix):

return s1.startswith(prefix) == prefix.startswith and s2.startswith(prefix) == prefix.startswith(prefix)

>>> both\_start\_with("sdfg","ser","s")

False

>>> def both\_start\_with(s1, s2, prefix):

print s1.startswith(prefix)

SyntaxError: invalid syntax

>>> def both\_start\_with(s1, s2, prefix):

print (s1.startswith(prefix))

>>> both\_start\_with("sdfg","ser","s")

True

>>> s1="sdf"

>>> print (s1.startswith)

<built-in method startswith of str object at 0x103d26490>

>>> def both\_start\_with(s1, s2, prefix):

if s1.startswith(prefix) and s2.startswith(prefix):

return True

else:

return False

>>> def both\_start\_with(s1, s2, prefix):

return s1.startswith(prefix) and s2.startswith(prefix)

>>> both\_start\_with("sdfg","ser","s")

True

>>> both\_start\_with("sdfg","ser","t")

False

>>> start = 'L'

>>> middle = 8

>>> end = 'R'

>>> start+str(middle)+end

'L8R'

>>> def larger\_of\_smallest(L1, L2):

return max(min(L1)) and min(L2)

>>> larger\_of\_smallest([1, 4, 0], [3, 2])

Traceback (most recent call last):

File "<pyshell#171>", line 1, in <module>

larger\_of\_smallest([1, 4, 0], [3, 2])

File "<pyshell#170>", line 2, in larger\_of\_smallest

return max(min(L1)) and min(L2)

TypeError: 'int' object is not iterable

>>> def larger\_of\_smallest(L1, L2):

return max(min(L1), min(L2))

>>> larger\_of\_smallest([1, 4, 0], [3, 2])

2

>>> larger\_of\_smallest([4], [9, 6, 3])

4

>>> def add\_to\_letter\_counts(d, s):

for c in s:

for c in d:

d[c]=d[c]+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

{'i': 5, 'r': 10, 'e': 6}

>>> def add\_to\_letter\_counts(d, s):

for c in s:

for d[c] in d:

d[c]=d[c]+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

Traceback (most recent call last):

File "<pyshell#188>", line 1, in <module>

add\_to\_letter\_counts(letter\_counts, 'eerie')

File "<pyshell#186>", line 4, in add\_to\_letter\_counts

d[c]=d[c]+1

TypeError: Can't convert 'int' object to str implicitly

>>> def add\_to\_letter\_counts(d, s):

for c in s:

for c in d:

d[c]=d[c]+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

{'i': 5, 'r': 10, 'e': 6}

>>> def add\_to\_letter\_counts(d, s):

for c in s:

for c in d:

d[c]+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

{'i': 0, 'r': 5, 'e': 1}

>>> def add\_to\_letter\_counts(d, s):

for c in s:

for c in d:

d[c]=d+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

Traceback (most recent call last):

File "<pyshell#200>", line 1, in <module>

add\_to\_letter\_counts(letter\_counts, 'eerie')

File "<pyshell#198>", line 4, in add\_to\_letter\_counts

d[c]=d+1

TypeError: unsupported operand type(s) for +: 'dict' and 'int'

>>> def add\_to\_letter\_counts(d, s):

for c in s:

if c in d[c]:

d[c]=d[c]+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

Traceback (most recent call last):

File "<pyshell#204>", line 1, in <module>

add\_to\_letter\_counts(letter\_counts, 'eerie')

File "<pyshell#202>", line 3, in add\_to\_letter\_counts

if c in d[c]:

TypeError: argument of type 'int' is not iterable

>>> def add\_to\_letter\_counts(d, s):

for c in s:

for c in d[c]:

d[c]=d[c]+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

Traceback (most recent call last):

File "<pyshell#208>", line 1, in <module>

add\_to\_letter\_counts(letter\_counts, 'eerie')

File "<pyshell#206>", line 3, in add\_to\_letter\_counts

for c in d[c]:

TypeError: 'int' object is not iterable

>>> def add\_to\_letter\_counts(d, s):

for c in s:

if c in d:

d[c]=d[c]+1

return d

>>> letter\_counts = {'i': 0, 'r': 5, 'e': 1}

>>> add\_to\_letter\_counts(letter\_counts, 'eerie')

{'i': 1, 'r': 6, 'e': 4}

>>>