CS 1358 Introduction to Programming in Python

Fall Semester 2019

Prof. Pai H. Chou

Self-Check 5

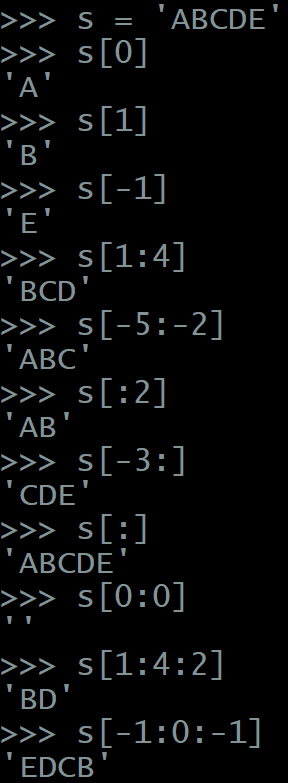
Due Date: Sunday, October 13, 2019, 11:59pm

Answer the following questions to check your understanding of your material. Expect the same kind of questions to show up on your tests.

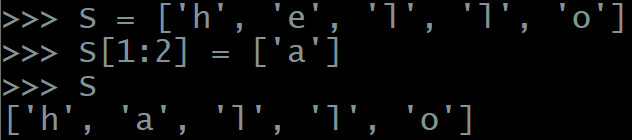
## 1. Definitions and Short Answers

1. What is the data type of (1, 2, 3)?

tuples

1. If s = 'ABCDE', what is the value of 
   * s[0] 'A'
   * s[1] 'B'
   * s[-1] 'E'
   * s[1:4] 'BCD'
   * s[-5:-2] 'ABC'
   * s[:2] 'AB'
   * s[-3:] 'CDE'
   * s[:] 'ABCDE'
   * s[0:0] ''
   * s[1:4:2] 'BD'
   * s[-1:0:-1] 'EDCB'
2. If S = ['h', 'e', 'l', 'l', 'o'], what is the value of S after executing the statement S[1:2] = ['a']?

S = ['h', 'a', 'l', 'l', 'o']

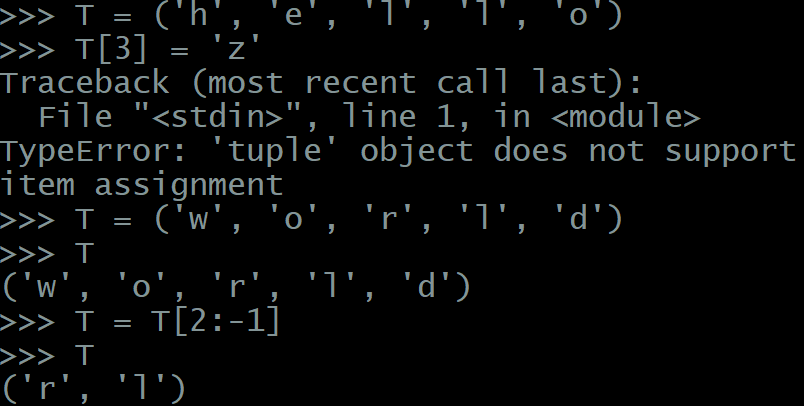


1. If T = ('h', 'e', 'l', 'l', 'o'), which of the following is allowed or not allowed and why?
   * T[3] = 'z'

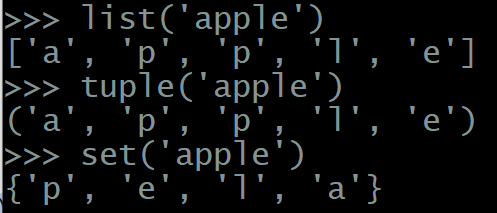
Not allowed, since T is a tuple, and a tuple is immutable.

* + T = ('w', 'o', 'r', 'l', 'd')

Allowed

* + T = T[2:-1]

Allowed

1. What is the value of
   * list('apple')

['a', 'p', 'p', 'l', 'e']

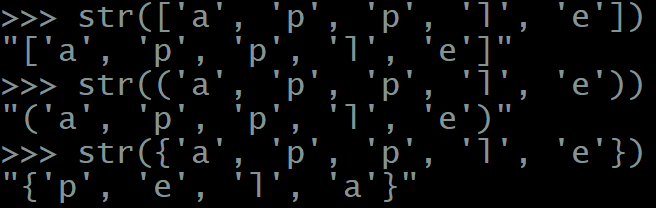
* + tuple('apple')

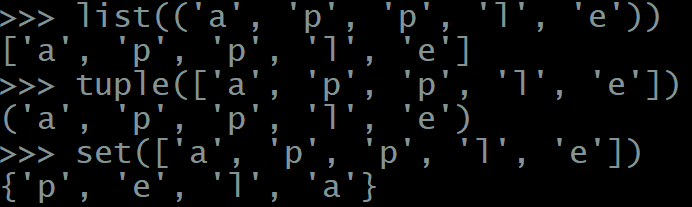
('a', 'p', 'p', 'l', 'e')

* + set('apple')

{'a', 'p', 'l', 'e'}

1. What is the value of
   * str(['a', 'p', 'p', 'l', 'e'])
   * str(('a', 'p', 'p', 'l', 'e'))
   * str({'a', 'p', 'p', 'l', 'e'})



1. What is the value of
   * list(('a', 'p', 'p', 'l', 'e'))

['a', 'p', 'p', 'l', 'e']

* + tuple(['a', 'p', 'p', 'l', 'e'])

('a', 'p', 'p', 'l', 'e')

* + set(['a', 'p', 'p', 'l', 'e'])

{'a', 'p', 'l', 'e'}

1. What is the result of
   * 'Apple' < 'apple'

True

* + 'Apple' <= 'apple'

True

* + 'Apple' == 'apple'

False

* + 'Apple' >= 'apple'

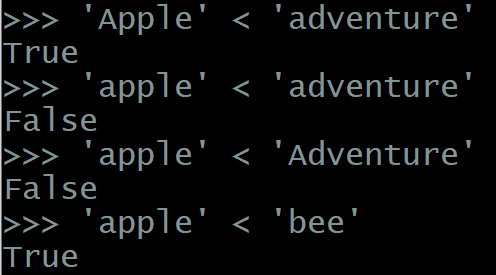
False

* + 'Apple' > 'apple'

False

* + 'Apple' != 'apple'

True

1. What is the result of
   * 'Apple' < 'adventure'

True

* + 'apple' < 'adventure'

False

* + 'apple' < 'Adventure'

False

* + 'apple' < 'bee'

True

* + 'apple' < 'Bee'

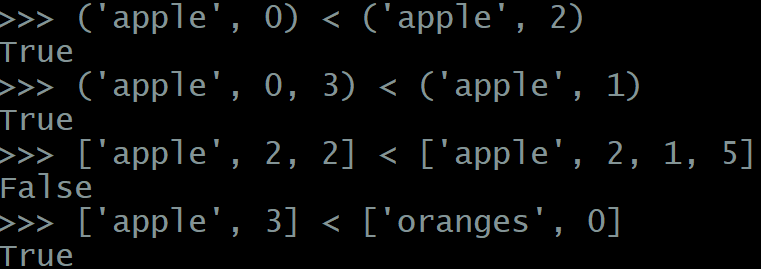
False

* + 'Apple' < 'bee'

True

* + 'Apple' < 'Bee'

True

1. What is the result of
   * ('apple', 0) < ('apple', 2)

True

* + ('apple', 0, 3) < ('apple', 1)

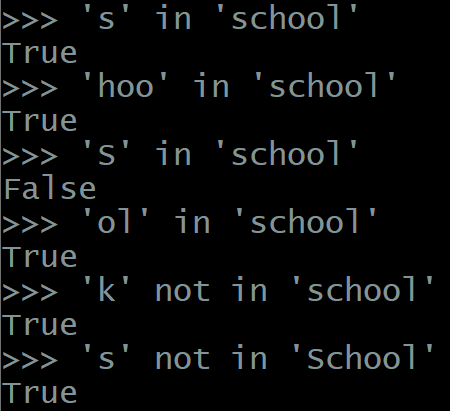
True

* + ['apple', 2, 2] < ['apple', 2, 1, 5]

False

* + ['apple', 3] < ['oranges', 0]

True

1. What is the result of
   * 's' in 'school'

True

* + 'hoo' in 'school'

True

* + 'S' in 'school'

False

* + 'ol' in 'school'

True

* + 'k' not in 'school'

True

* + 's' not in 'School'

True

1. What is the result of
   * 's' in ['s', 'c', 'h', 'o', 'o', 'l']

True

* + ['s'] in ['s', 'c', 'h', 'o', 'o', 'l']

False

* + ['s'] in [['s'], ['c'], ['h'], ['o'], ['o'], ['l']]

True

* + 'hoo' in ['s', 'c', 'h', 'o', 'o', 'l']

False

* + ['h', 'o', 'o'] in ['s', 'c', 'h', 'o', 'o', 'l']

False

* + ('h', 'o', 'o') in ['s', 'c', ('h', 'o', 'o'), 'l']

True

* + ('h', 'o', 'o') not in ('s', 'c', ('h', 'o', 'o'), 'l')

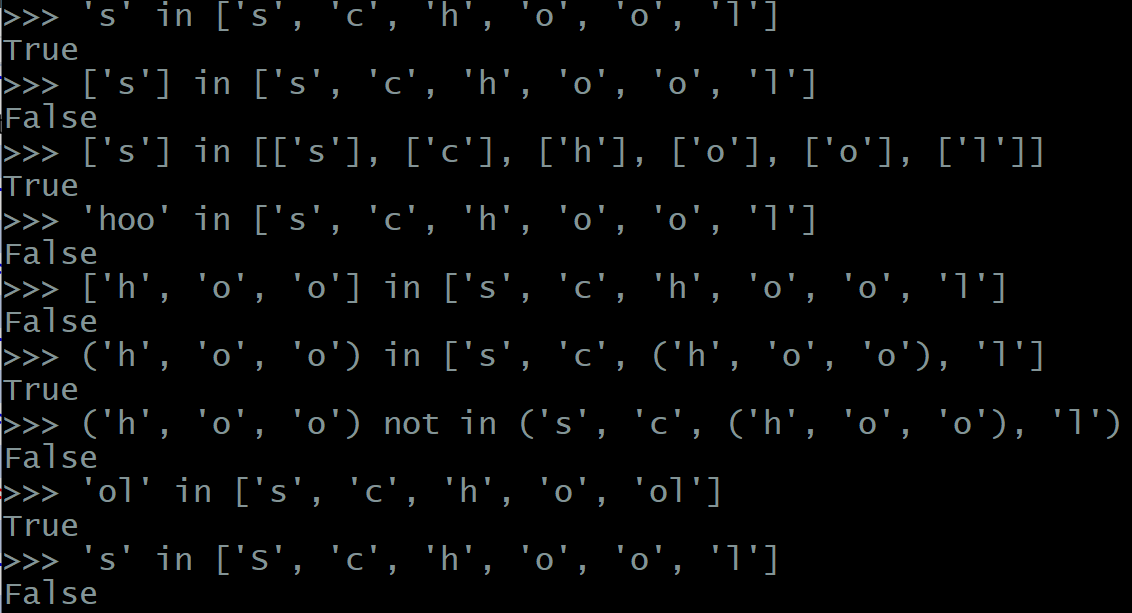
False

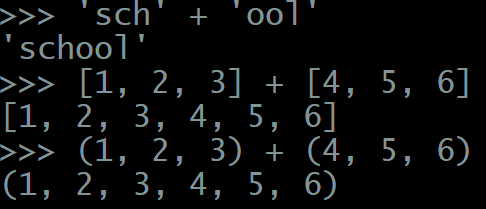
* + 'ol' in ['s', 'c', 'h', 'o', 'ol']

True

* + 's' in ['S', 'c', 'h', 'o', 'o', 'l']

False



1. What is the result of
   * 'sch' + 'ool'

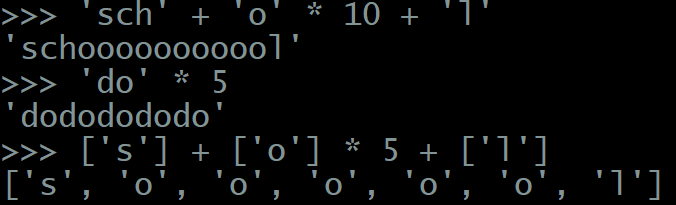
'school'

* + [1, 2, 3] + [4, 5, 6]

[1, 2, 3, 4, 5, 6]

* + (1, 2, 3) + (4, 5, 6)

(1, 2, 3, 4, 5, 6)

1. What is the result of
   * 'sch' + 'o' \* 10 + 'l'

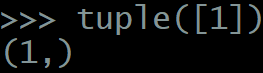
'schooooooooool'

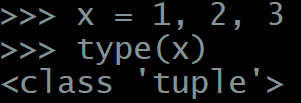
* + 'do' \* 5

'dododododo'

* + ['s'] + ['o'] \* 5 + ['l']

['s', 'o', 'o', 'o', 'o', 'o', 'l']

1. How do you express a tuple literal of a single element? For example, how do you write a tuple literal that has the same value as tuple([1])?

(1,)

1. Suppose you have x = 1, 2, 3

What is the value of type(x)?

tuple

1. Suppose you have L = ['f', 'r', 'o', 'g']  
   What is the new value of L after executing each of the following statements in order?
   * L.append('s')

['f', 'r', 'o', 'g', 's']

* + L.extend(['p', 'o', 'n', 'd'])

['f', 'r', 'o', 'g', 's', 'p', 'o', 'n', 'd']

* + L.insert(4, ' ')

['f', 'r', 'o', 'g', ' ', 's', 'p', 'o', 'n', 'd']

* + L.reverse()

['d', 'n', 'o', 'p', 's', ' ', 'g', 'o', 'r', 'f']

* + L.sort()

[' ', 'd', 'f', 'g', 'n', 'o', 'o', 'p', 'r', 's']

* + L.remove('o')

[' ', 'd', 'f', 'g', 'n', 'o', 'p', 'r', 's']

* + L.pop()

[' ', 'd', 'f', 'g', 'n', 'o', 'p', 'r']

* + L.pop(0)

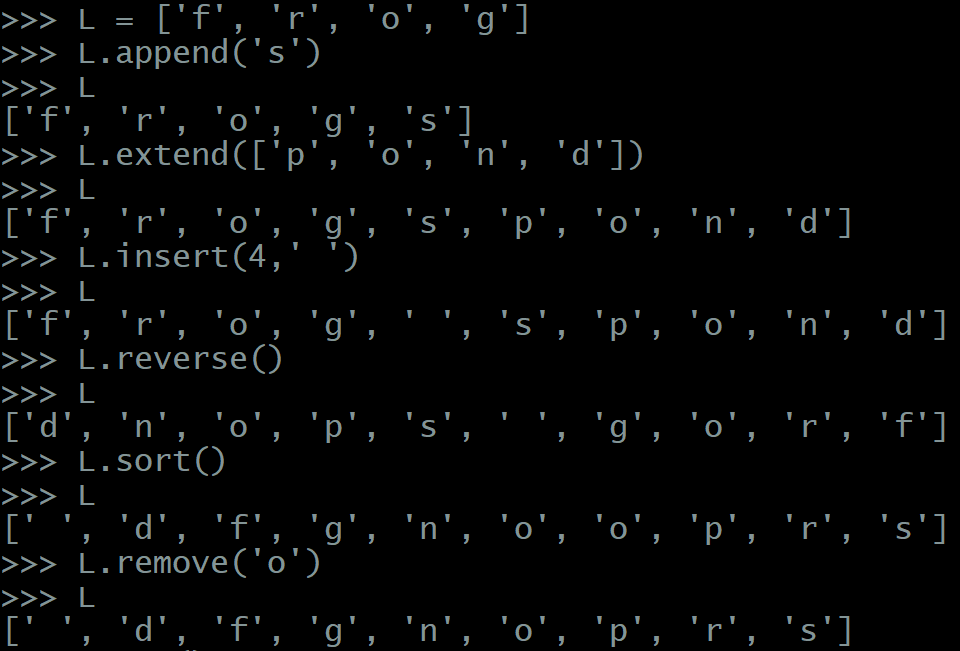
['d', 'f', 'g', 'n', 'o', 'p', 'r']

* + L.clear()

[]

* + L.append('z')

['z']





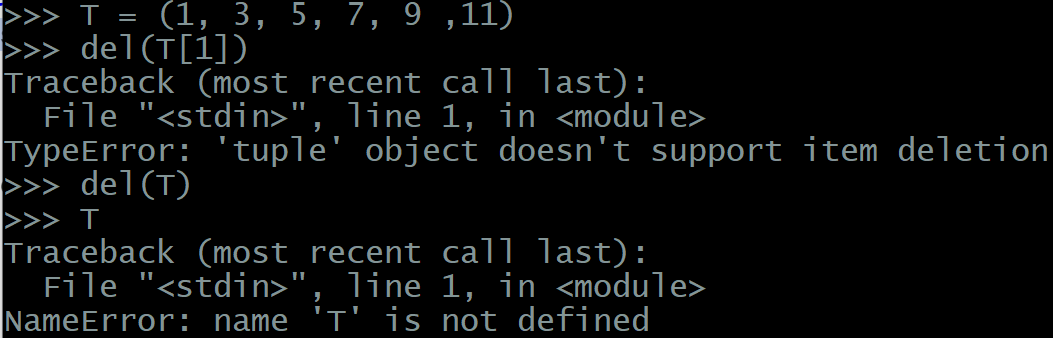
1. If T = (1, 3, 5, 7, 9, 11), Can you call del(T[1])? why or why not?

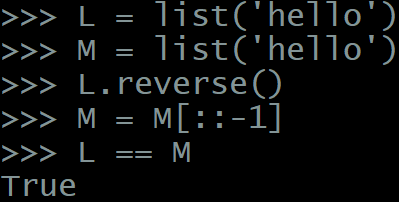
No, you can’t. ~~Since T is a tuple, and a tuple is mutable.~~

‘tuple’ doesn’t support item deletion

Can you call del(T)? What is the effect?

Yes, delete the tuple T



1. Suppose L = list('hello') and separately M = list('hello'). After executing  
   L.reverse() L = ['o', 'l', 'l, 'e', 'h']   
   M = M[::-1] M = ['o', 'l', 'l, 'e', 'h']
   * is L == M evaluate to True or False?

True

* + What is the difference between these two ways of reversing elements in a list?

L.reverse() does not create a new list

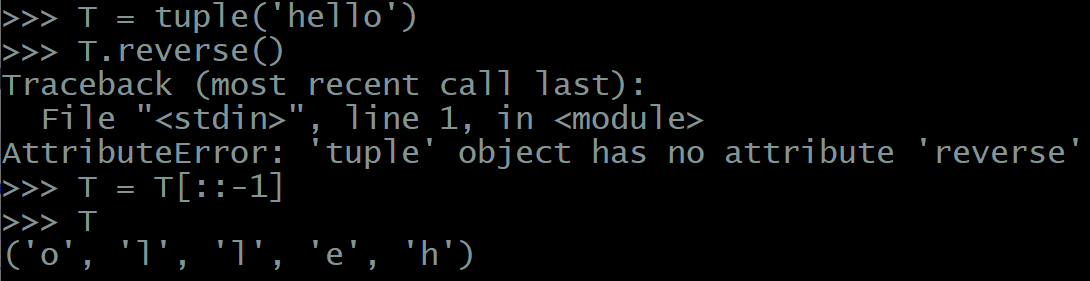
M = M[::-1] let M refers to a newly created list

1. if T = tuple('hello'), are the following statements allowed in Python? Why or why not?
   * T.reverse()

No, since the tuple is immutable.

* + T = T[::-1]

Yes, since this command create a new tuple



1. What is a **stack** as a data structure?

last-in, first-out (LIFO) data structure

What is another name (4-letter initialism) for a stack?

LIFO

How can a stack be implemented using a list?

Show how **push** and **pop** can be accomplished by calling list methods.

push -> append()

pop -> pop()

1. What is a **queue** as a data structure?

first-in, first-out(FIFO) data structure

What is another name (4-letter initialism) for a queue?

FIFO

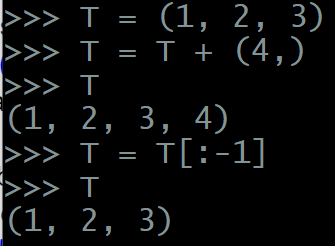
How can a queue be implemented using a list?

Show how enqueue and dequeue can be accomplished by calling list methods.

enqueue -> append()

dequeue -> pop(0)

1. Show how a **tuple** can be used to implement
   * a stack's push and pop functionality



* + a queue's enqueue and dequeue functionality



* + Is a tuple more or less efficient than a list for implementing the stack and queue data structures? Why?

less efficient.

Since list is mutable, just using the list function could do the jobs, but since tuple is immutable, we have to implement the “Non-mutation” method to do so.

1. What do these built-in functions do?
   * max(['h', 'e', 'l', 'l', 'o'])

'o'

* + min('hello')

'e'

* + sum([2, 3, 4, 5, 6])

20

* + sum(range(10))

45 \*range(10) 是從0~9，不包含4

* + any(['', 'apples', 'oranges', 'banana'])

True

* + any([0, '', 0.0, [], ()])

False

* + any(['0', '', 0.0, [], ()])

True

* + any([0, ' ', 0.0, [], ()])

True

* + all(['', 'apples', 'oranges', 'banana'])

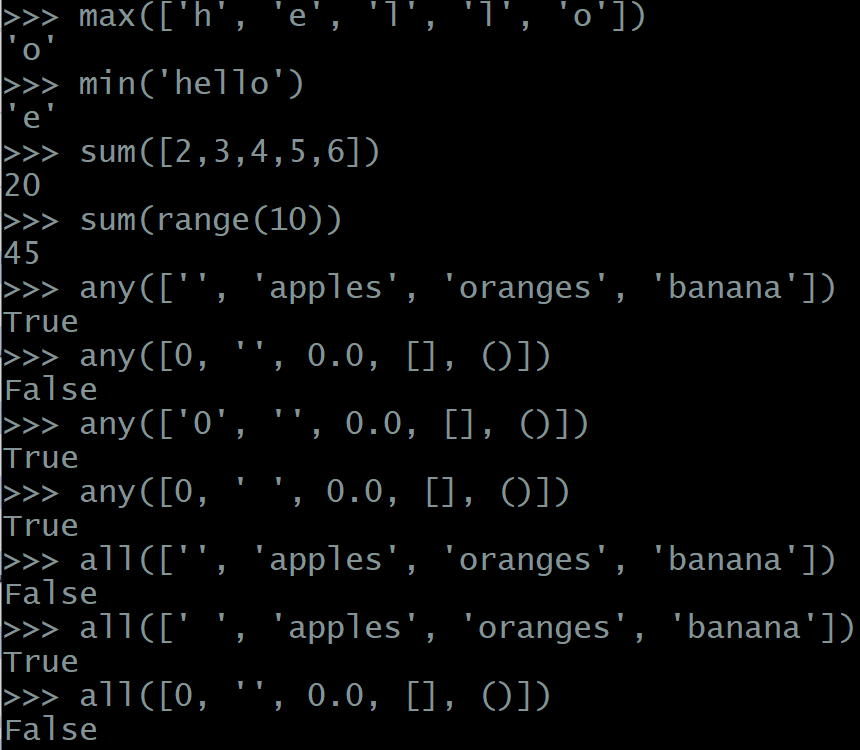
False

* + all([' ', 'apples', 'oranges', 'bananas'])

True

* + all([0, '', 0.0, [], ()])

False



1. What is the **non-mutation** version of the following statements? Assume L is a list
   * L.sort()

L = sorted(S)

* + L.reverse()

S = list(reversed(S))

S = S[::-1]

* + L.extend([1, 2, 3])

S = S + [1,2,3]

* + del(L[1])

S = S[:1] + S[2:]

* + L.pop()

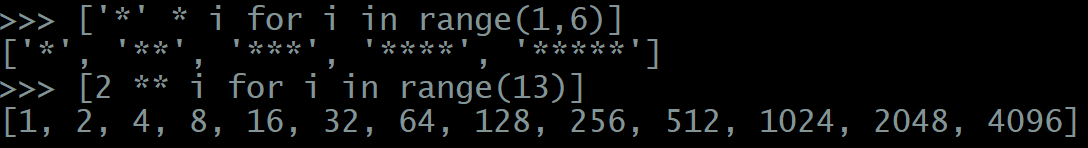
S = S[:-1]

1. How do you use **list comprehension** to create a list with values
   * ['\*', '\*\*', '\*\*\*', '\*\*\*\*', '\*\*\*\*\*']

['\*' \* i for i in range(1,6)]

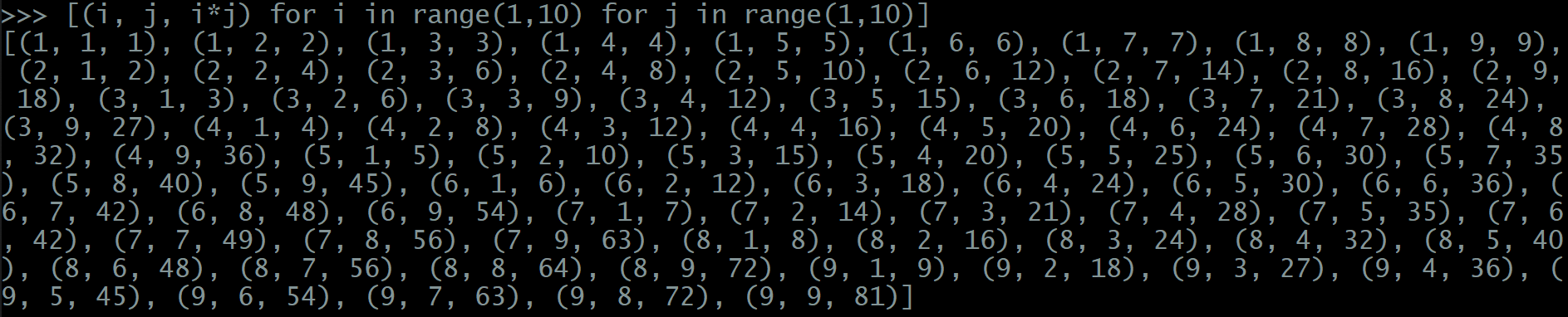
* + [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]

[2 \*\* i for i in range(13)]



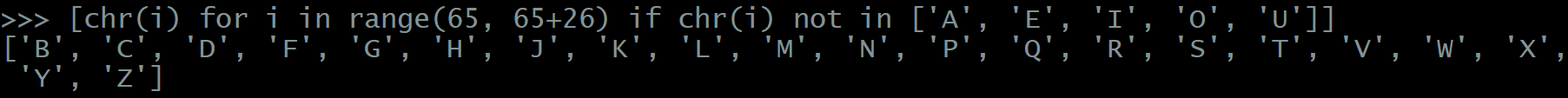
1. How do you use **two-level list comprehension** to create a multiplication table in the following format: [(1, 1, 1), (1, 2, 2), (1, 3, 3), … (1, 9, 9), (2, 1, 2), (2, 2, 4), (2, 3, 6), (2, 4, 8), … (2, 9, 18), (3, 1, 3), (3, 2, 6), (3, 3, 9), … (3, 9, 27), (4, 1, 4) … (4, 9, 36), (5, 1, 5), … (9, 9, 81)]

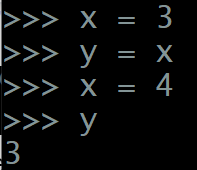
[(i, j, i\*j) for i in range(1,10) for j in range(1,10)]



1. How do you use list comprehension with filter to generate the list of upper-case letters except 'A', 'E', 'I', 'O', 'U'?

[chr(i) for i in range(65, 65+26) if chr(i) not in ['A', 'E', 'I', 'O', 'U']]



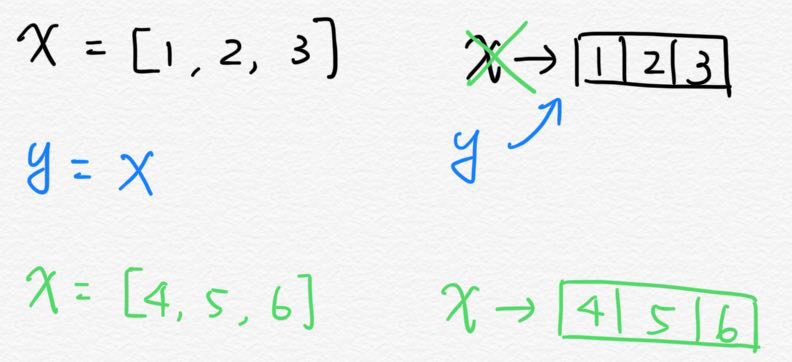
1. After executing the following sequence of statements:  
   x = 3

y = x

x = 4

what is the value of y?

3

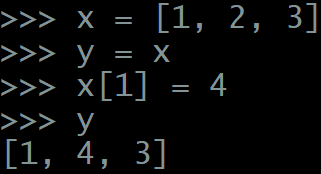
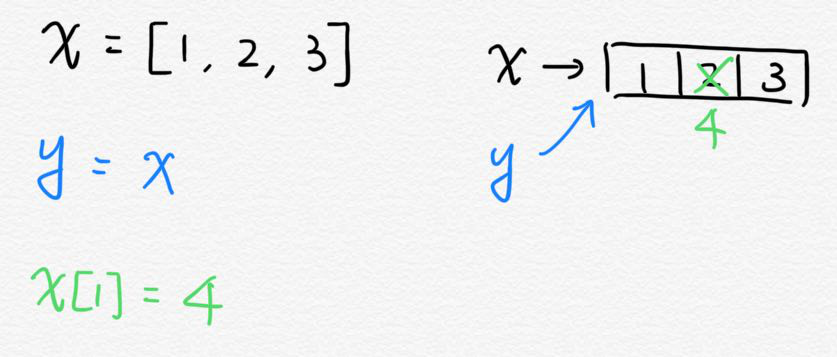
1. After executing the following sequence of statements  
   x = [1, 2, 3]

y = x

x = [4, 5, 6]

what is the value of y?

y = [1, 2, 3]

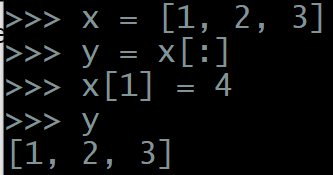
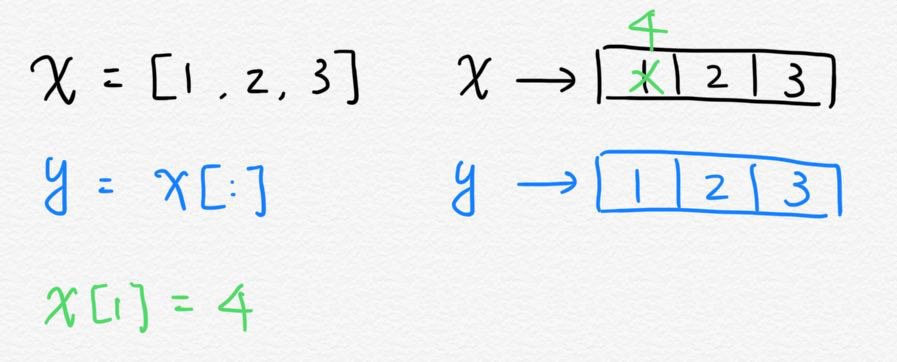
1. After executing the following sequence of statements  
   x = [1, 2, 3]

**y = x**

x[1] = 4

what is the value of y?

y = [1, 4, 3]

1. After executing the following sequence of statements  
   x = [1, 2, 3]

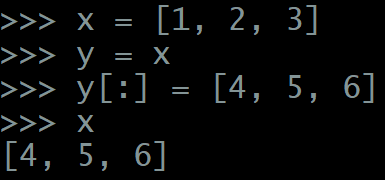
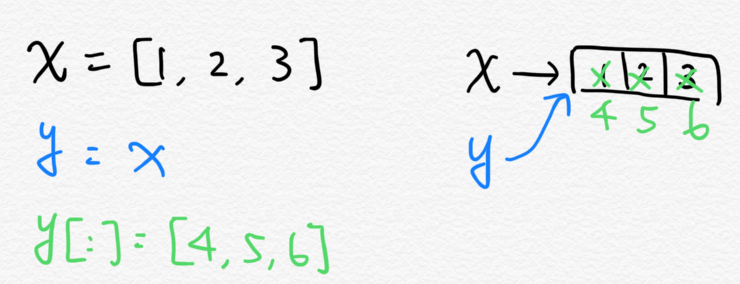
y = **x[:]**

x[1] = 4

what is the value of y?

y = [1, 2, 3]

\* y = x[:] is shallow copy

1. After executing the following sequence of statements  
   x = [1, 2, 3]

y = x

**y[:] =** [4, 5, 6]

what is the value of x?

x = [4, 5, 6]

\* y = x is shallow copy

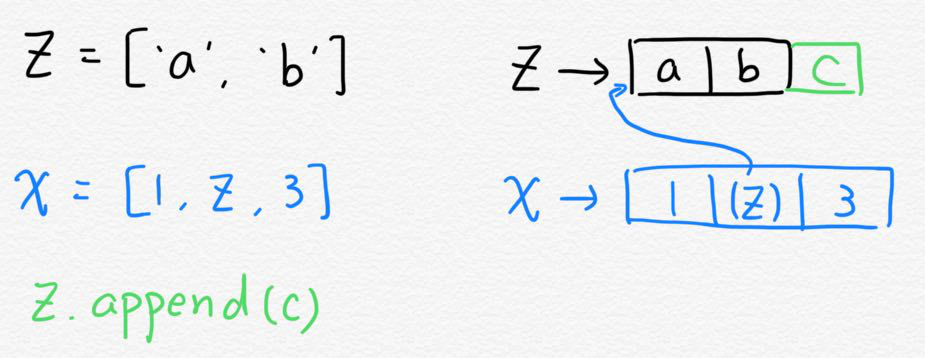
1. After executing the following sequence of statements  
   z = ['a', 'b']

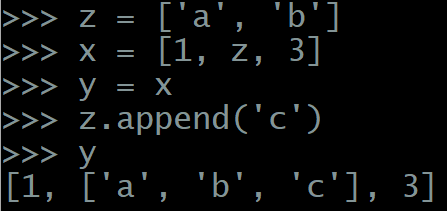
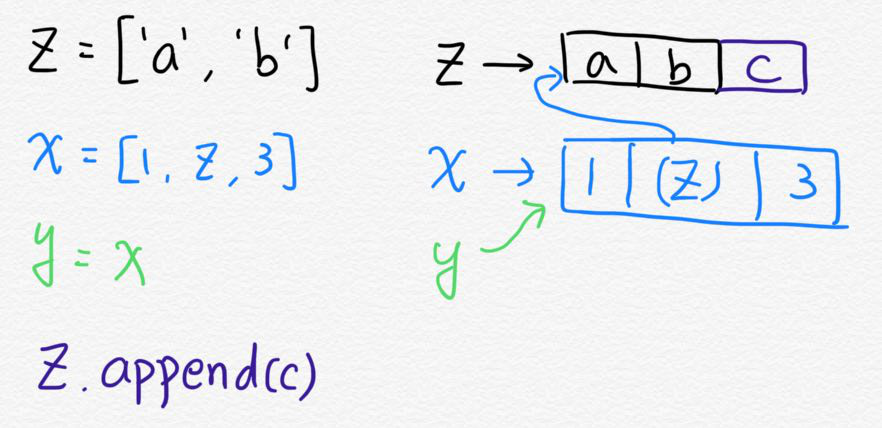
**x = [1, z, 3]**

z.append('c')

what is the value of x?

x = [1, ['a', 'b', 'c'], 3]



1. After executing the following sequence of statements  
   z = ['a', 'b']

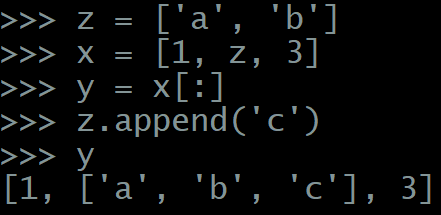
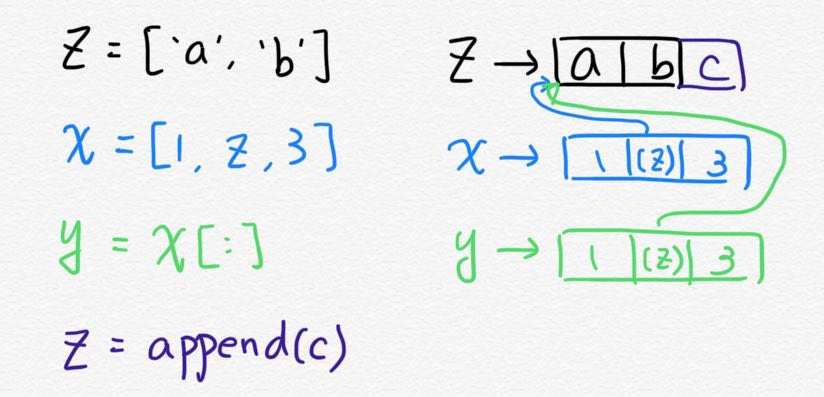
x = [1, z, 3]

**y = x**

z.append('c')

what is the value of y?

y = [1, ['a', 'b', 'c'], 3]

1. After executing the following sequence of statements  
   z = ['a', 'b']

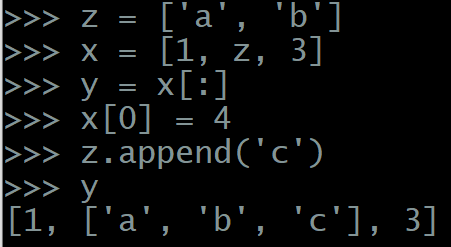
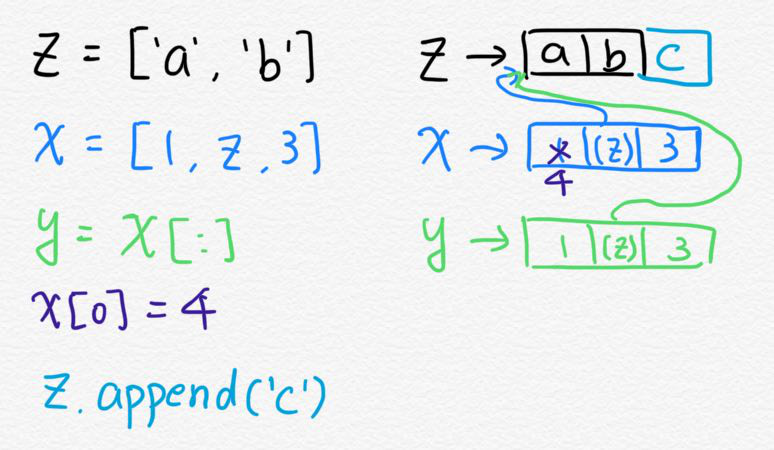
x = [1, z, 3]

**y = x[:]**

z.append('c')

what is the value of y?

y = [1, ['a', 'b', 'c'], 3]

1. After executing the following sequence of statements  
   z = ['a', 'b']

x = [1, z, 3]

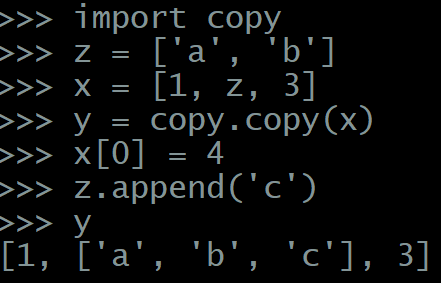
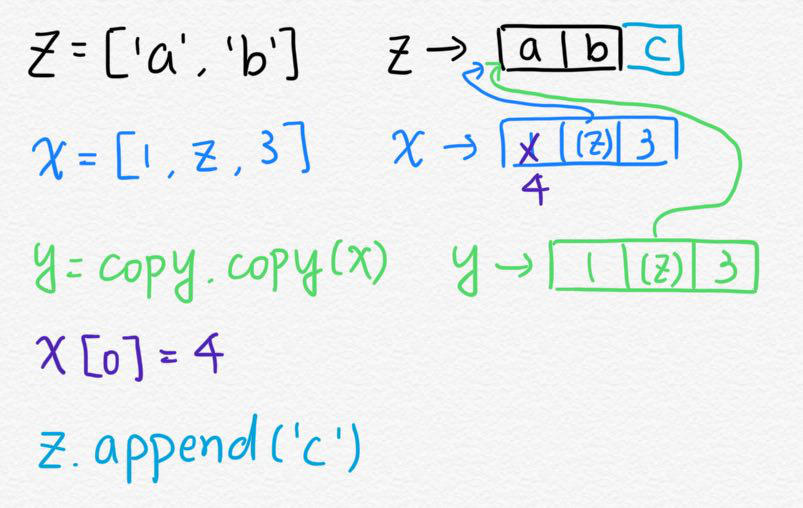
y = **x[:]**

**x[0] =** 4

z.append('c')

what is the value of y?

y = [1, ['a', 'b', 'c'], 3]

1. After executing the following sequence of statements  
   import copy

z = ['a', 'b']

x = [1, z, 3]

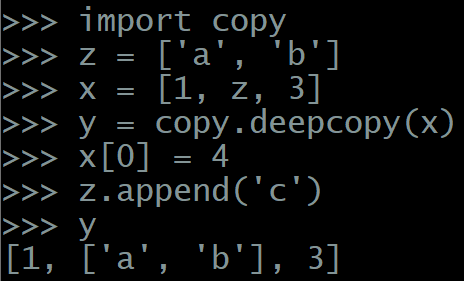
y = **copy.copy**(x)

x[0] = 4

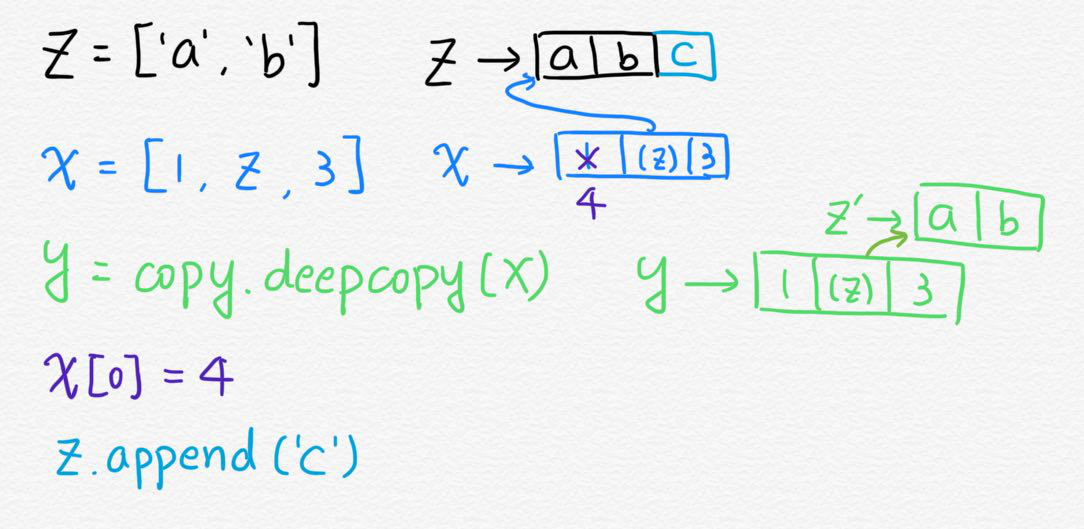
z.append('c')

what is the value of y?

y = [1, ['a', 'b', 'c'], 3]

1. After executing the following sequence of statements  
   import copy

z = ['a', 'b']

x = [1, z, 3]

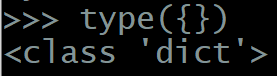
y = **copy.deepcopy**(x)

x[0] = 4

z.append('c')

what is the value of y?

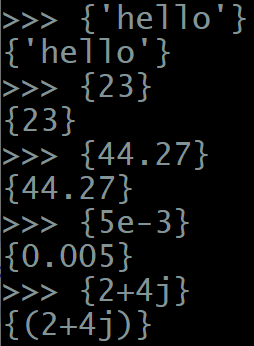
y = [1, ['a', 'b'], 3]

1. What is the **type** of {}?

dict

1. What is the expression for an **empty set**?

set()

1. Which of the following can or cannot be a **member of a set**? Why?
   * 'hello'

Legal

* + 23

Legal

* + 44.27

Legal

* + 5e-3

Legal

* + 2+4j

Legal

* + ['Mary', 'had', 'a', 'little', 'lamb']

Not legal, since list is mutable

* + ('Mary', 'had', 'a', 'little', 'lamb')

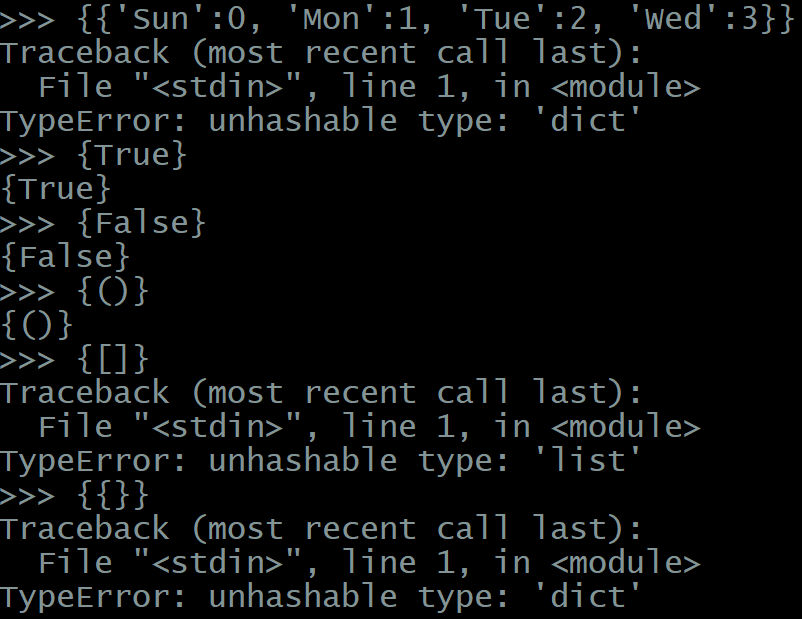
Legal

* + {'Mary', 'had', 'a', 'little', 'lamb'}

Not legal, since set is mutable



* + {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3}

Not legal, since dict is mutable

* + True

Legal

* + False

Legal

* + ()

Legal

* + []

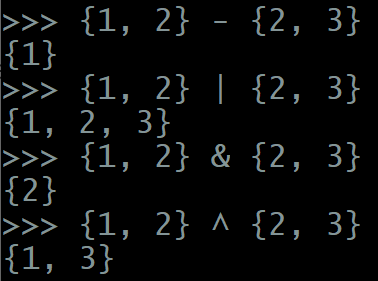
Not legal, since list is mutable

* + {}

Not legal, since dict is mutable

1. What is the value of len(set('hello'))?

4

1. What is the value of each of the following expressions?
   * {1, 2} - {2, 3}

{1}

* + {1, 2} | {2, 3}

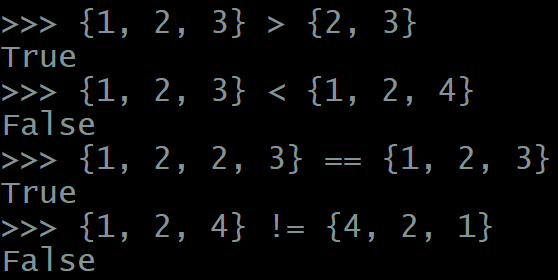
{1, 2, 3}

* + {1, 2} & {2, 3}

{2}

* + {1, 2} ^ {2, 3}

{1,3}

1. What is the result of the following comparisons?
   * {1, 2, 3} > {2, 3}

True

* + {1, 2, 3} < {1, 2, 4}

False

* + {1, 2, 2, 3} == {1, 2, 3}

True

* + {1, 2, 4} != {4, 2, 1}

False

1. Assume S = {1, 2, 3}, what is the difference between  
   S = S | {3, 4} and S |= {3, 4}?

S = S | {3, 4} S | {3, 4} is a new set, and S refers to a newly created set

S |= {3, 4} S |= {3, 4} modify the set S itself

1. Assume D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3}
   * What is the value of D['Mon']?

1

* + What is the value of D after D['Thu'] = 4?

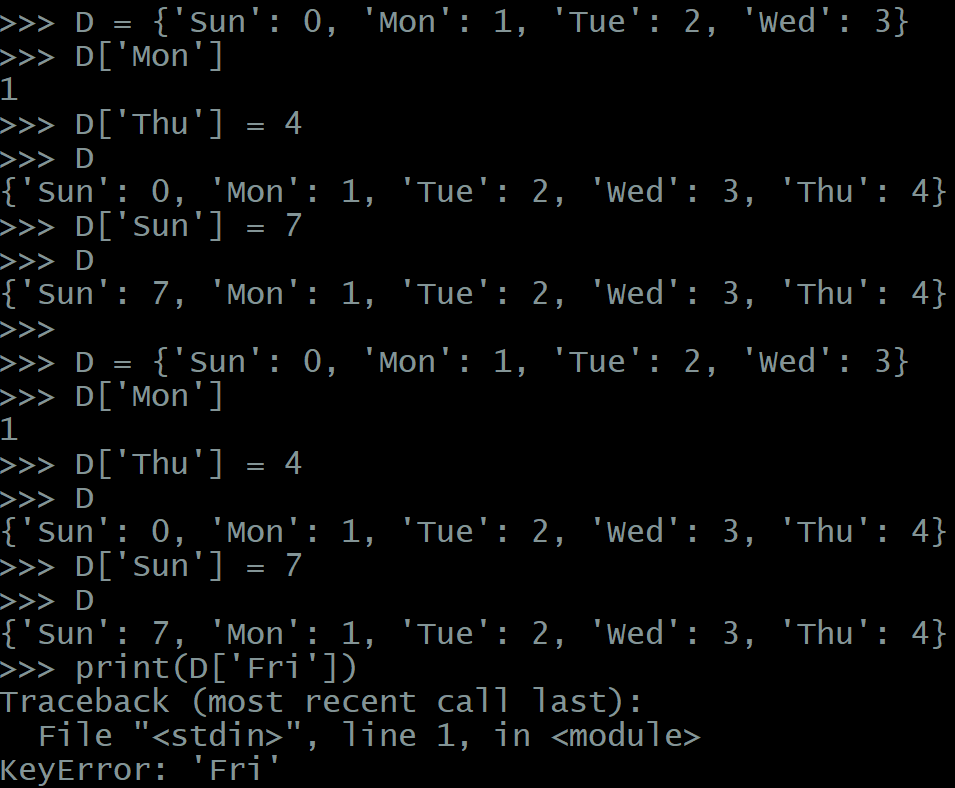
D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3, 'Thu': 4}

* + Continuing with the previous statement, what is the value of D after D['Sun'] = 7?

D = {'Sun': 7, 'Mon': 1, 'Tue': 2, 'Wed': 3, 'Thu': 4}

* + What happens if you attempt print(D['Fri'])?

Keyerror



1. Assume D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3}
   * What is the value of D.keys()

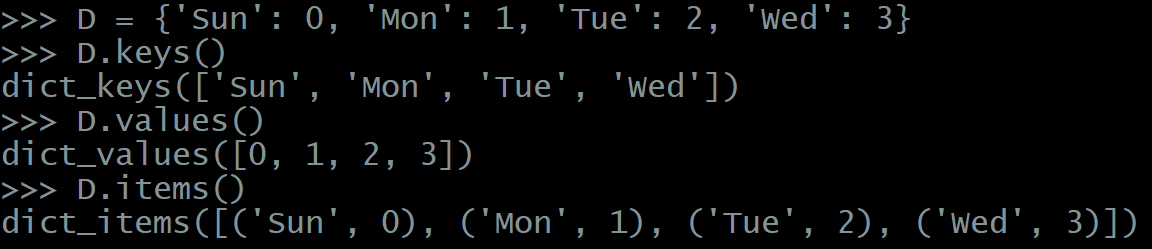
dict\_keys(['Sun', 'Mon', 'Tue', 'Wed'])

* + What is the value of D.values()

dict\_values([0, 1, 2, 3])

* + What is the value of D.items()

dict\_items([('Sun', 0), ('Mon', 1), ('Tue', 2), ('Wed', 3)])



1. Assuming D = {}, which of the following is legal or not legal in Python? If not legal, why not?
   * D[()] = 10

Legal

* + D[''] = {}

Legal

* + D[0] = ''

Legal

* + D[{}] = ()

Not legal, since {} is a dictionary and it’s mutable → can’t be the key

* + D[[]] = set()

Not legal, since [] is a list, and the list is mutable → can’t be the key

* + D[:] = range(10)

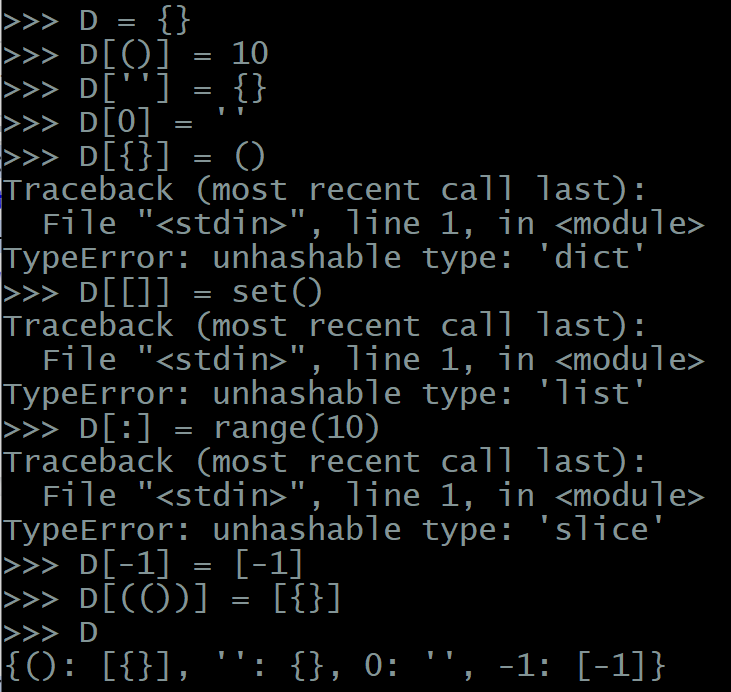
Not legal, since slice is unhashable

* + D[-1] = [-1]

Legal

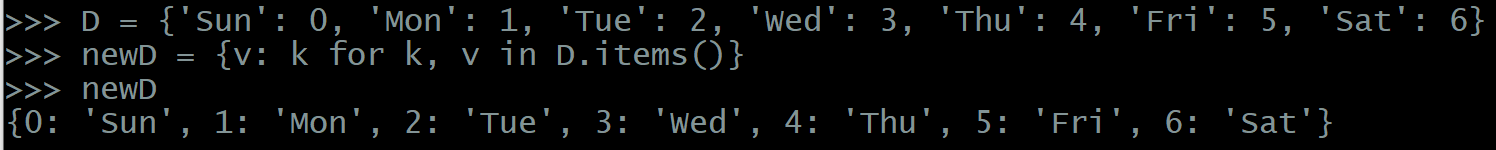
* + D[(())] = [{}]

Legal, since (()) is a tuple

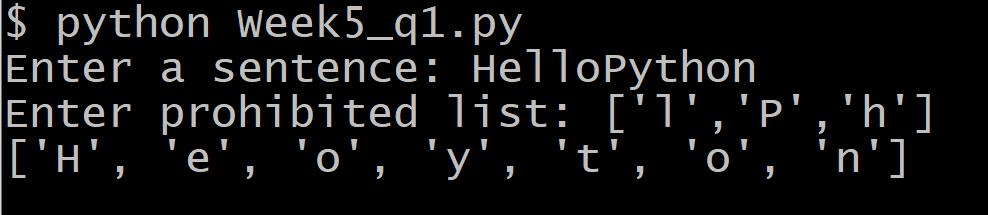


1. How do you use dictionary comprehension to create a reverse mapping? For example, suppose D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3, 'Thu': 4, 'Fri': 5, 'Sat': 6}, create its reverse mapping whose value should be {0: 'Sun', 1: 'Mon', 2: 'Tue', 3: 'Wed', 4: 'Thu', 5: 'Fri', 6: 'Sat'}?

newD = { v: k for k, v in D.items()}



## 2. Programming Exercises

1. Write a program that prompts the user to input a sentence and a prohibited list. The program needs to report a list which includes all characters in the sentence except those appear in the prohibited list. For example, (blue text = typed input, green highlight = program printout)  
   $ python3 Week5\_q1.py  
   Enter a sentence: HelloPython  
   Enter prohibited list: ['l','P','h']  
   ['H', 'e', 'o', 'y', 't', 'o', 'n']  
   

Note : Sentence is case sensitive.

