**DATA STRUCTURE**

**5.1. More on lists**

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

>>> fruits.count('apple')

2

>>> fruits.count('tangerine')

0

>>> fruits.index('banana')

3

>>> fruits.index('banana', 4) # Find next banana starting a position 4

6

>>> fruits.reverse()

>>> fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']

>>> fruits.append('grape')

>>> fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']

>>> fruits.sort()

>>> fruits

['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']

>>> fruits.pop()

'pear'

**5.1.1. Using Lists as Stacks**

>>> stack = [3, 4, 5]

>>> stack.append(6)

>>> stack.append(7)

>>> stack

[3, 4, 5, 6, 7]

>>> stack.pop()

7

>>> stack

[3, 4, 5, 6]

>>> stack.pop()

6

>>> stack.pop()

5

>>> stack

[3, 4]

>>>

**5.1.2. Using Lists as Queues**

>>> from collections import deque

File "<stdin>", line 1

from collections import deque

^

IndentationError: unexpected indent

>>> from collections import deque

>>> queue = deque(["Eric", "John", "Michael"])

>>> queue.append("Terry") # Terry arrives

>>> queue.append("Graham") # Graham arrives

>>> queue.popleft() # The first to arrive now leaves

'Eric'

>>> queue.popleft() # The second to arrive now leaves

'John'

>>> queue # Remaining queue in order of arrival

deque(['Michael', 'Terry', 'Graham'])

**5.1.3. List Comprehensions**

>>> squares = []

File "<stdin>", line 1

squares = []

^

IndentationError: unexpected indent

>>> squares = []

>>> for x in range(10):

... squares.append(x\*\*2)

...

>>> squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>>

>>> combs = []

>>> for x in [1,2,3]:

... for y in [3,1,4]:

... if x != y:

... combs.append((x, y))

...

>>> combs

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> combs = []

>>> for x in [1,2,3]:

... for y in [3,1,4]:

... if x != y:

... combs.append((x, y))

...

>>> combs

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

>>> vec = [-4, -2, 0, 2, 4]

>>> # create a new list with the values doubled

... [x\*2 for x in vec]

[-8, -4, 0, 4, 8]

>>> # filter the list to exclude negative numbers

... [x for x in vec if x >= 0]

[0, 2, 4]

>>>

>>> # apply a function to all the elements

... [abs(x) for x in vec]

[4, 2, 0, 2, 4]

>>> # call a method on each element

... freshfruit = [' banana', ' loganberry ', 'passion fruit ']

>>> [weapon.strip() for weapon in freshfruit]

['banana', 'loganberry', 'passion fruit']

>>> # create a list of 2-tuples like (number, square)

... [(x, x\*\*2) for x in range(6)]

[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

>>> # the tuple must be parenthesized, otherwise an error is raised

... [x, x\*\*2 for x in range(6)]

File "<stdin>", line 2

[x, x\*\*2 for x in range(6)]

^

SyntaxError: invalid syntax

>>>

>>> # flatten a list using a listcomp with two 'for'

... vec = [[1,2,3], [4,5,6], [7,8,9]]

>>> [num for elem in vec for num in elem]

[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> from math import pi

>>> [str(round(pi, i)) for i in range(1, 6)]

['3.1', '3.14', '3.142', '3.1416', '3.14159']

**5.1.4. Nested List Comprehensions**

>>> matrix = [

... [1, 2, 3, 4],

... [5, 6, 7, 8],

... [9, 10, 11, 12],

... ]

>>> [[row[i] for row in matrix] for i in range(4)]

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

>>> transposed = []

>>> for i in range(4):

... transposed.append([row[i] for row in matrix])

...

>>> transposed

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

>>> transposed = []

>>> for i in range(4):

... # the following 3 lines implement the nested listcomp

... transposed\_row = []

... for row in matrix:

... transposed\_row.append(row[i])

... transposed.append(transposed\_row)

...

>>> transposed

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

>>> list(zip(\*matrix))

[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]

**5.2. The del statement**

>>> a = [-1, 1, 66.25, 333, 333, 1234.5]

>>> del a[0]

>>> a

[1, 66.25, 333, 333, 1234.5]

>>> del a[2:4]

>>> a

[1, 66.25, 1234.5]

>>> del a[:]

>>> a

[]

>>> del a

**5.3. Tuples and Sequences**

>>> t = 12345, 54321, 'hello!'

>>> t[0]

12345

>>> t

(12345, 54321, 'hello!')

>>> # Tuples may be nested:

... u = t, (1, 2, 3, 4, 5)

>>> u

((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))

>>> # Tuples are immutable:

... t[0] = 88888

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

TypeError: 'tuple' object does not support item assignment

>>> # but they can contain mutable objects:

... v = ([1, 2, 3], [3, 2, 1])

>>> v

([1, 2, 3], [3, 2, 1])

>>> empty = ()

>>>

>>> singleton = 'hello', # <-- note trailing comma

>>> len(empty)

0

>>> len(singleton)

1

>>>

>>> singleton

('hello',)

>>> x, y, z = t

**5.4. Sets**

>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

>>> print(basket) # show that duplicates have been removed

{'apple', 'pear', 'orange', 'banana'}

>>> 'orange' in basket # fast membership testing

True

>>> 'crabgrass' in basket

False

>>> # Demonstrate set operations on unique letters from two words

...

>>> a = set('abracadabra')

>>> b = set('alacazam')

>>> a # unique letters in a

{'a', 'c', 'b', 'd', 'r'}

>>> a - b # letters in a but not in b

{'d', 'r', 'b'}

>>> a | b # letters in a or b or both

{'a', 'z', 'm', 'c', 'b', 'l', 'd', 'r'}

>>> a & b # letters in both a and b {

{'a', 'c'}

>>> a ^ b # letters in a or b but not both

{'b', 'r', 'z', 'm', 'd', 'l'}

>>>

>>> a = {x for x in 'abracadabra' if x not in 'abc'}

>>> a

{'d', 'r'}

**5.5. Dictionaries**

>>> tel = {'jack': 4098, 'sape': 4139}

>>> tel['guido'] = 4127

>>> tel

{'jack': 4098, 'sape': 4139, 'guido': 4127}

>>> tel['jack']

4098

>>> del tel['sape']

>>> tel['irv'] = 4127

>>> tel

{'jack': 4098, 'guido': 4127, 'irv': 4127}

>>> list(tel)

['jack', 'guido', 'irv']

>>> sorted(tel)

['guido', 'irv', 'jack']

>>> 'guido' in tel

True

>>> 'jack' not in tel

False

>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])

{'sape': 4139, 'guido': 4127, 'jack': 4098}

>>> {x: x\*\*2 for x in (2, 4, 6)}

{2: 4, 4: 16, 6: 36}

>>> dict(sape=4139, guido=4127, jack=4098)

{'sape': 4139, 'guido': 4127, 'jack': 4098}

**5.6. Looping Techniques**

>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}

>>> for k, v in knights.items():

... print(k, v)

...

gallahad the pure

robin the brave

>>> for i, v in enumerate(['tic', 'tac', 'toe']):

... print(i, v)

...

0 tic

1 tac

2 toe

>>> questions = ['name', 'quest', 'favorite color']

>>> answers = ['lancelot', 'the holy grail', 'blue']

>>> for q, a in zip(questions, answers):

... print('What is your {0}? It is {1}.'.format(q, a))

...

What is your name? It is lancelot.

What is your quest? It is the holy grail.

What is your favorite color? It is blue.

>>> for i in reversed(range(1, 10, 2)):

... print(i)

...

9

7

5

3

1

>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

>>> for f in sorted(set(basket)):

... print(f)

...

apple

banana

orange

pear

>>> import math

>>> raw\_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]

>>> filtered\_data = []

>>> for value in raw\_data:

... if not math.isnan(value):

... filtered\_data.append(value)

...

>>> filtered\_data

[56.2, 51.7, 55.3, 52.5, 47.8]

>>>

**5.7. More on Conditions**

>>> string1, string2, string3 = '', 'Trondheim', 'Hammer Dance'

>>> non\_null = string1 or string2 or string3

>>> non\_null

'Trondheim'