CS677 - Data Science with Python

Python Essentials - Review

Functions

In [1]:
```
def greet(name, className):
    print("Hello " + name + ", Welcome to " + className)
```

In [2]:
```
greet('John', 'CS677')
```
```
Hello John, Welcome to CS677
```

In [3]:
```
greet('Jane', 'CS521')
```
```
Hello Jane, Welcome to CS521
```

In [4]:
```
print(greet('Jane', 'CS521'))
```
```
Hello Jane, Welcome to CS521
None
```

In [5]:
```
def greet(name, className):
    return ("Hello " + name + ", Welcome to " + className)
```

In [6]:
```
print(greet('Jane', 'CS521'))
```
```
Hello Jane, Welcome to CS521
```

Strings

In [7]:

```
s1 = 'programming'
s2 = "programming"
s3 = '''program
    ming'''
print(s1)
print(s2)
print(s3)
```

```
programming
programming
program
    ming
```

In [8]:

```
print(len(s1))
print(len(s2))
print(len(s3))
```

```
11
11
16
```

Strings - Indexing and slicing

In [9]:

```
s1 = 'programming'
print(s1)

print(s1[0])
print(s1[0:1])

print(s1[-1])

print(s1[0:7])
print(s1[0:-4])
print(s1[:7])

print(s1[3:-4])
print(s1[3:7])

print(s1[3:])
```

```
programming
p
p
g
program
program
program
gram
gram
gramming
```

Strings - Skip slicing

In [10]:
```
s1 = 'abcdefghijk'
print(s1)

print(s1[::2])

print(s1[:7:2])
print(s1[4::2])

print(s1[::-1])
print(s1[::-2])
```

```
abcdefghijk
acegik
aceg
egik
kjihgfedcba
kigeca
```

Strings are Immutable

In [11]:
```
print(s1)
s1[0] = 'A'
```

```
abcdefghijk

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-9177bd4911ec> in <module>
      1 print(s1)
----> 2 s1[0] = 'A'

TypeError: 'str' object does not support item assignment
```

Strings - Concatenation Operators + and *

In [12]:

```
s1 = 'Data'
s2 = 'Science'
print(s1 + s2)
print(s1 + ' ' + s2)

print(2 * s1)
print(s2 * 3)

print(2 * (s1 + s2))
```

```
DataScience
Data Science
DataData
ScienceScienceScience
DataScienceDataScience
```

Strings - membership

In [13]:

```
vowels = 'aeiou'

print('p' in vowels)
print('e' in vowels)
print('t' not in vowels)
```

```
False
True
True
```

In [14]:

```
print('log' in 'hologram')
print('hal' in 'hologram')
```

```
True
False
```

Strings - Iterating

In [15]:
```
s = 'Python'

for letter in s:
    print(letter, ord(letter))
```

```
P 80
y 121
t 116
h 104
o 111
n 110
```

Strings - other methods

In [16]:

```
s = 'PyThon is Fun'
print(s)
print("capitalize:", s.capitalize())
print("lower:", s.lower())
print("upper:", s.upper())
print("title:", s.title())
print("swapcase:", s.swapcase())
print("split:", s.split())
print("list:", list(s))
print("find:", s.find('is'))
print("find:", s.find('n'))
print("find:", s.find('n', 6))
print("find:", s.find('n', -1))
print("find:", s.find('a'))
```

```
PyThon is Fun
capitalize: Python is fun
lower: python is fun
upper: PYTHON IS FUN
title: Python Is Fun
swapcase: pYtHON IS fUN
split: ['PyThon', 'is', 'Fun']
list: ['P', 'y', 'T', 'h', 'o', 'n', ' ', 'i', 's', ' ', 'F', 'u', 'n']
find: 7
find: 5
find: 12
find: 12
find: -1
```

Strings - formatting

In [17]:
```
s = "Hello {}, your grade is {}.".format("John", 85)
print(s)

s = "Hello {:10s}, your grade is {:10.2f}.".format("John", 85)
print(s)

s = "Hello {:>10s}, your grade is {:<10.2f}.".format("John", 85)
print(s)

s = "Hello {:^10s}, your grade is {:^10.2f}.".format("John", 85)
print(s)
```
```
Hello John, your grade is 85.
Hello John      , your grade is      85.00.
Hello       John, your grade is 85.00     .
Hello    John   , your grade is   85.00   .
```

Out of range index

In [18]:
```
s = 'Hello'
s[6]
```
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-18-4cfe130d0688> in <module>
      1 s = 'Hello'
----> 2 s[6]

IndexError: string index out of range
```

Lists

- like arrays in other languages
- access to elements is O(1)

In [19]:
```
l1 = []
l2 = [10, 20, 30]
l3 = ['Python', 75, 'Java', 60]
l4 = [[10, 20], [30, 40, 50]]

print(l1)
print(l2)
print(l3)
print(l4)

print(len(l1))
print(len(l2))
print(len(l3))
print(len(l4))
```

```
[]
[10, 20, 30]
['Python', 75, 'Java', 60]
[[10, 20], [30, 40, 50]]
0
3
4
2
```

Lists - Indexing

In [20]:
```
data = ['Python', 75, 'Java', 60]

print(data[0])
print(data[3])

print(data[-1])
print(data[-4])
```
```
Python
60
60
Python
```

Lists - Slicing

In [21]:
```
data = ['Python', 75, 'Java', 60, 'R', 90]

print(data)
print(data[0:4])
print(data[2:])
print(data[:4])
print(data[-2:])
print(data[-4:-2])
```
```
['Python', 75, 'Java', 60, 'R', 90]
['Python', 75, 'Java', 60]
['Java', 60, 'R', 90]
['Python', 75, 'Java', 60]
['R', 90]
['Java', 60]
```

Lists - Skip slicing

In [22]:
```
data = ['Python', 75, 'Java', 60, 'R', 90]

print(data)
print(data[::2])
print(data[1::2])
print(data[:4:2])

print(data[::-1])
print(data[::-2])
```

```
['Python', 75, 'Java', 60, 'R', 90]
['Python', 'Java', 'R']
[75, 60, 90]
['Python', 'Java']
[90, 'R', 60, 'Java', 75, 'Python']
[90, 60, 75]
```

Modifying a list

In [23]:
```
data = ['Python', 75, 'Java', 60, 'R', 90]
print(data)

data[1] = 99
print(data)

data[1::2] = [66,77,88]
print(data)

data.extend(['NumPy', 55])
print(data)

data.append('Pandas')
print(data)

data.insert(-1, 'TensorFlow')
print(data)

data.insert(-2, 'Keras')
print(data)
```

```
['Python', 75, 'Java', 60, 'R', 90]
['Python', 99, 'Java', 60, 'R', 90]
['Python', 66, 'Java', 77, 'R', 88]
['Python', 66, 'Java', 77, 'R', 88, 'NumPy', 55]
['Python', 66, 'Java', 77, 'R', 88, 'NumPy', 55, 'Pandas']
['Python', 66, 'Java', 77, 'R', 88, 'NumPy', 55, 'TensorFlow', 'Pandas']
['Python', 66, 'Java', 77, 'R', 88, 'NumPy', 55, 'Keras', 'TensorFlow', 'Pand
as']
```

In [24]:
```
data = ['Python', 75]

data.append('Java')
print(data)

data[len(data):] = [60]
print(data)
```

```
['Python', 75, 'Java']
['Python', 75, 'Java', 60]
```

Concatenation of Lists with operators + and *

In [25]:
```
skills = ['Python', 'Java', 'R']
levels  = [75, 85, 60]

result = skills + levels
print(result)

print(levels + skills)

print(skills + ['Numpy'])

print(2 * skills)

print(2 * levels)

print(3 * levels)
```

```
['Python', 'Java', 'R', 75, 85, 60]
[75, 85, 60, 'Python', 'Java', 'R']
['Python', 'Java', 'R', 'Numpy']
['Python', 'Java', 'R', 'Python', 'Java', 'R']
[75, 85, 60, 75, 85, 60]
[75, 85, 60, 75, 85, 60, 75, 85, 60]
```

Removing items from list

In [26]:
```
data = ['Python', 75, 'Java', 60, 'R', 90]
print(data)

del data[1:3]
print(data)
```

```
['Python', 75, 'Java', 60, 'R', 90]
['Python', 60, 'R', 90]
```

In [27]:
```
data = ['Python', 75, 'Java', 60, 'R', 90]
print(data)

data.remove('Java')
print(data)

print(data.pop())
print(data)

print(data.pop(2)) # remove at specified index
print(data)

data.clear()
print(data)
```

```
['Python', 75, 'Java', 60, 'R', 90]
['Python', 75, 60, 'R', 90]
90
['Python', 75, 60, 'R']
60
['Python', 75, 'R']
[]
```

Other List methods

In [28]:

```
data = [10, 15, 20, 15, 7, 15, 40, 15]
print(data)

print(data.count(15))
print(data.index(15))
print(data.index(15, 2))
```

```
[10, 15, 20, 15, 7, 15, 40, 15]
4
1
3
```

In [29]:

```
data = [10, 20, 15, 7, 40, 15]
print(data)

data.sort()
print(data)

data.reverse()
print(data)

data = [10, 20, 50, 100]
data.sort(reverse = True)
print(data)
```

```
[10, 20, 15, 7, 40, 15]
[7, 10, 15, 15, 20, 40]
[40, 20, 15, 15, 10, 7]
[100, 50, 20, 10]
```

In [30]:
```
data = [10, 20, 15, 7, 40, 15]
print(sorted(data))

print(data)

print(data.sort())

print(data)
```

```
[7, 10, 15, 15, 20, 40]
[10, 20, 15, 7, 40, 15]
None
[7, 10, 15, 15, 20, 40]
```

In [31]:
```
data1 = [10, 20, 30]
data2 = data1
data2[0] = 100

print(data1)
print(data2)
```

```
[100, 20, 30]
[100, 20, 30]
```

In [32]:
```
data1 = [10, 20, 30]
data2 = data1.copy()   # Shallow copy
data2[0] = 100

print(data1)
print(data2)
```

```
[10, 20, 30]
[100, 20, 30]
```

In [33]:
```
data1 = [10, 20, 30]
data2 = data1[:]        # Shallow copy
data2[0] = 100

print(data1)
print(data2)
```

```
[10, 20, 30]
[100, 20, 30]
```

membership in a list

In [34]:
```
data = ['Python', 75, 'Java', 60, 'R', 90]
print(data)

print('Python' in data)
print('NumPy' in data)
print(60 in data)
print(70 not in data)
```

```
['Python', 75, 'Java', 60, 'R', 90]
True
False
True
True
```

range

- **return an immutable sequence of numbers**

range(stop) - range of numbers from 0 to $stop - 1$

```
In [35]: print(range(10))
```

```
range(0, 10)
```

```
In [36]: print(list(range(10)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

range(start, stop) - range of numbers from start to $stop - 1$

```
In [37]: print(list(range(0, 10)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [38]: print(list(range(10, 20)))
```

```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
In [39]: print(list(range(20, 10)))
```

```
[]
```

range(start, stop, step)

```
In [40]: print(list(range(10, 20, 2)))
```

```
[10, 12, 14, 16, 18]
```

```
In [41]: print(list(range(10, 20, -2)))
```

```
[]
```

In [42]:
```
print(list(range(20, 10, -2)))
```
```
[20, 18, 16, 14, 12]
```

In [43]:
```
print(list(range(20, 10, -1)))
```
```
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11]
```

Iterating using range

In [44]:
```
for i in range(10, 20):
    print('Square of', i, 'is', i*i)
```
```
Square of 10 is 100
Square of 11 is 121
Square of 12 is 144
Square of 13 is 169
Square of 14 is 196
Square of 15 is 225
Square of 16 is 256
Square of 17 is 289
Square of 18 is 324
Square of 19 is 361
```

In [45]:
```
skills = ['Python', 'Java', 'R']

for i in range(len(skills)):
    print('Skill#{} is {}'.format(i+1, skills[i]))
```
```
Skill#1 is Python
Skill#2 is Java
Skill#3 is R
```

Iterating over lists

In [46]:
```
skills = ['Python', 'Java', 'R']
levels  = [75, 85, 60]

for skill in skills:
    print(skill)
```

```
Python
Java
R
```

In [47]:
```
for index, skill in enumerate(skills):
    print(skill, '-->', levels[index])
```

```
Python --> 75
Java --> 85
R --> 60
```

zip - returns an iterator of tuples

In [48]:
```
print(list(zip(skills, levels)))
```

```
[('Python', 75), ('Java', 85), ('R', 60)]
```

In [49]:
```
for skill, level in zip(skills, levels):
    print(skill, '-->', level)
```

```
Python --> 75
Java --> 85
R --> 60
```

unzip

In [50]:
```python
data = [('Python', 75), ('Java', 85), ('R', 60)]

skills, levels = zip(*data)
print(skills)
print(levels)
```

```
('Python', 'Java', 'R')
(75, 85, 60)
```

out of range index

In [51]:
```python
skills = ['Python', 'Java', 'R']
skills[3]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-51-85142c8d05c3> in <module>
      1 skills = ['Python', 'Java', 'R']
----> 2 skills[3]

IndexError: list index out of range
```

Tuples

- a sequence of values separated by a comma

In [52]:
```
x = (10, 20, 30)
print(x)
print(len(x))

y = ('Alice', (10, 20), [30, 40], (50, 60))
print(y)
print(len(y))

z = 100, 'Python', 80, 'R'
print(z)
```

```
(10, 20, 30)
3
('Alice', (10, 20), [30, 40], (50, 60))
4
(100, 'Python', 80, 'R')
```

Tuple Indexing and Slicing - similar to list

In [53]:
```
print(x)
print(x[0], x[-1])

print(y)
print(y[1:3])
print(y[::-1])
```

```
(10, 20, 30)
10 30
('Alice', (10, 20), [30, 40], (50, 60))
((10, 20), [30, 40])
((50, 60), [30, 40], (10, 20), 'Alice')
```

Concatenation and repetition

In [54]:
```
print(x + y)
```
```
(10, 20, 30, 'Alice', (10, 20), [30, 40], (50, 60))
```

In [55]:
```
print(x * 3)
print(2 * y)
```
```
(10, 20, 30, 10, 20, 30, 10, 20, 30)
('Alice', (10, 20), [30, 40], (50, 60), 'Alice', (10, 20), [30, 40], (50, 6
0))
```

count and index

In [56]:
```
z = x * 3
print(z)
```
```
(10, 20, 30, 10, 20, 30, 10, 20, 30)
```

In [57]:
```
z.count(30)
```
Out[57]:
```
3
```

In [58]:
```
z.count(40)
```
Out[58]:
```
0
```

In [59]:
```
z.index(30)
```
Out[59]:
```
2
```

In [60]:
```
z.index(40)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-60-0c24aa2375fa> in <module>
----> 1 z.index(40)

ValueError: tuple.index(x): x not in tuple
```

membership test

In [61]:
```
z = 100, 'Python', 80, 'R'
print(z)

print(80 in z)
print('Python' in z)
print('Java' not in z)
```

```
(100, 'Python', 80, 'R')
True
True
True
```

Iterating over a tuple

In [62]:
```
x = (10, 20, 30)

for value in x:
    print(value, value*value)
```

```
10 100
20 400
30 900
```

```
In [63]: for index, value in enumerate(x):
             print(index, value)

0 10
1 20
2 30
```

other built-in functions with tuple

```
In [64]: x = (10, 20, 30)
         y = ('R', 'Java', 'Python')

In [65]: (min(x), max(x), sum(x))

Out[65]: (10, 30, 60)

In [66]: min(y), max(y)

Out[66]: ('Java', 'R')

In [67]: sorted(x, reverse = True)

Out[67]: [30, 20, 10]

In [68]: sorted(y)

Out[68]: ['Java', 'Python', 'R']

In [69]: all(x)

Out[69]: True
```

In [70]:

```
y = (True, False, True)

print(all(y))
print(any(y))
```

```
False
True
```

Lists versus Tuples

In [71]:

```
list_values  = [10, 20, 30, 40]
tuple_values = (10, 20, 30, 40)

print(list_values)
print(tuple_values)
```

```
[10, 20, 30, 40]
(10, 20, 30, 40)
```

In [72]:

```
print(len(list_values))
print(len(tuple_values))
```

```
4
4
```

In [73]:

```
print(list_values.__sizeof__())
print(tuple_values.__sizeof__())
```

```
72
56
```

In [74]:
```
# Mutable versus Immutable

list_values[-1] = 400
print(list_values)
```
```
[10, 20, 30, 400]
```

In [75]:
```
tuple_values[-1] = 400
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-31d12efbfa96> in <module>
----> 1 tuple_values[-1] = 400

TypeError: 'tuple' object does not support item assignment
```

In [76]:
```
list_values.append(50)
print(list_values)
```
```
[10, 20, 30, 400, 50]
```

In [77]:
```
tuple_values.append(50)
```
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-77-821316f2950c> in <module>
----> 1 tuple_values.append(50)

AttributeError: 'tuple' object has no attribute 'append'
```

In [78]:
```
y = ('Alice', (10, 20), [30, 40], (50, 60))
y[2][0] = 300
print(y)
```
```
('Alice', (10, 20), [300, 40], (50, 60))
```

Lists and List Comprehensions

In [79]:
```
# Using loop

word = 'Data Science'
letters = []

for letter in word:
    letters.append(letter)

print(letters)
```
```
['D', 'a', 't', 'a', ' ', 'S', 'c', 'i', 'e', 'n', 'c', 'e']
```

In [80]:
```
# Using list comprehension

word = 'Data Science'

letters = [letter for letter in word]

print(letters)
```
```
['D', 'a', 't', 'a', ' ', 'S', 'c', 'i', 'e', 'n', 'c', 'e']
```

In [81]:
```
# Using Lambda function

word = 'Data Science'

letters = list(map(lambda x: x, word))

print(letters)
```
```
['D', 'a', 't', 'a', ' ', 'S', 'c', 'i', 'e', 'n', 'c', 'e']
```

In [82]:
```
# Conditions in List Comprehension

word = 'Data Science'

letters = [letter for letter in word if letter not in 'aeiou']

print(letters)
```
```
['D', 't', ' ', 'S', 'c', 'n', 'c']
```

In [83]:
```
data = [70, 30, 50, 80, 40, 90]
result = [score for score in data if score > 50]
print(result)
```
```
[70, 80, 90]
```

In [84]:
```
# if-else

word = 'Data Science'

vowel_or_consonant = ['V' if letter in 'aeiou' else 'C' for letter in word]

print(vowel_or_consonant)
```
```
['C', 'V', 'C', 'V', 'C', 'C', 'C', 'V', 'V', 'C', 'C', 'V']
```

Nested Lists

In [85]:
```
# Using loops

result = []
for i in range(1,6):
    new_row = []
    for j in range(1,6):
        new_row.append((i,j))
    result.append(new_row)

print(result)
```
```
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(2, 1), (2, 2), (2, 3), (2, 4),
(2, 5)], [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5)], [(4, 1), (4, 2), (4, 3),
(4, 4), (4, 5)], [(5, 1), (5, 2), (5, 3), (5, 4), (5, 5)]]
```

In [86]:
```
# Using list comprehension

[[(i,j) for j in range(1, 6)] for i in range(1, 6)]
```

Out[86]:
```
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)],
 [(2, 1), (2, 2), (2, 3), (2, 4), (2, 5)],
 [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5)],
 [(4, 1), (4, 2), (4, 3), (4, 4), (4, 5)],
 [(5, 1), (5, 2), (5, 3), (5, 4), (5, 5)]]
```

In [87]:
```
[[(i,j) for j in range(1, i+1)] for i in range(1, 6)]
```

Out[87]:
```
[[(1, 1)],
 [(2, 1), (2, 2)],
 [(3, 1), (3, 2), (3, 3)],
 [(4, 1), (4, 2), (4, 3), (4, 4)],
 [(5, 1), (5, 2), (5, 3), (5, 4), (5, 5)]]
```

In [88]:
```
# flatten list

data = [[1,2,3], [4], [5,6]]
[val for row in data for val in row]
```

Out[88]:
```
[1, 2, 3, 4, 5, 6]
```

In [ ]:
```

```

In [89]:
```
# print 2D lists

def print2D(x):
    for row in x:
        for value in row:
            print('{:4}'.format(value),  end = '')
        print()
```

In [90]:
```
# Nested loops - matrix transpose using loops

data = [[1,2,3], [4,5,6]]
print2D(data)

result = []

for col in range(len(data[0])):
    new_row = []

    for row in data:
        new_row.append(row[col])

    result.append(new_row)

print("Transpose is", result)
print2D(result)
```

```
   1   2   3
   4   5   6
Transpose is [[1, 4], [2, 5], [3, 6]]
   1   4
   2   5
   3   6
```

In [91]:
```
# Using list comprehension

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

result = [ [row[col] for row in data] for col in range(len(data[0]))]

print(result)
```

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

In [92]:
```
# Using list comprehension

def print2D(x):
    print('\n'.join([''.join(['{:4}'.format(item) for item in row])
      for row in x]))

print2D(data)

print("Transpose is")
print2D(result)
```

```
   1   2   3
   4   5   6
Transpose is
   1   4
   2   5
   3   6
```

In [93]:
```
print2D([[i*j for j in range(1, 11)] for i in range(1, 11)])
```

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

Dictionary

- unordered collection of key-value pairs
- indexed by keys
    - can be any immutable type - strings, numbers, tuples with only strings and numbers
- empty dictionary - {}

In [94]:
```
d1 = {}
d2 = {1: 'Python', 2: 'Java'}
d3 = {'Java': ['CS520', 'CS526'], 'Python': ['CS521', 'CS677', 'CS767'] }

print(d1)
print(d2)
print(d3)
```

```
{}
{1: 'Python', 2: 'Java'}
{'Java': ['CS520', 'CS526'], 'Python': ['CS521', 'CS677', 'CS767']}
```

The dict() constructor builds dictionaries directly from sequences of key-value pairs

In [95]:
```
d2 = dict({1: 'Python', 2: 'Java'})
d3 = dict([ ('Java', ['CS520', 'CS526']), ('Python', ['CS521', 'CS677', 'CS76
7'])])

print(d2)
print(d3)
```

```
{1: 'Python', 2: 'Java'}
{'Java': ['CS520', 'CS526'], 'Python': ['CS521', 'CS677', 'CS767']}
```

In [96]:
```
print(d2[1])
print(d3['Python'])
print(d3[d2[1]])
```

```
Python
['CS521', 'CS677', 'CS767']
['CS521', 'CS677', 'CS767']
```

In [97]:
```
d3['R']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-97-1860c03d820e> in <module>
----> 1 d3['R']

KeyError: 'R'
```

In [98]:
```
print(d2.get(1))
print(d3.get('Python'))
print(d3.get(d2.get(1)))
```

```
Python
['CS521', 'CS677', 'CS767']
['CS521', 'CS677', 'CS767']
```

In [99]:
```
print(d3.get('R'))
```

```
None
```

Alternative ways of creating dictionaries

In [100]:
```
a = dict(one=1, two=2, three=3)
b = {'one': 1, 'two': 2, 'three': 3}
c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
d = dict([('two', 2), ('one', 1), ('three', 3)])
e = dict({'three': 3, 'one': 1, 'two': 2})

print(a)

print(a == b == c == d == e)
```

```
{'one': 1, 'two': 2, 'three': 3}
True
```

Modifying and adding items in a dictionary

In [101]:
```
d2 = dict({1: 'Python', 2: 'Java'})
print(d2)

d2[2] = 'C++'
d2[5] = 'R'
d2['a'] = 'Go'

print(d2)
```

```
{1: 'Python', 2: 'Java'}
{1: 'Python', 2: 'C++', 5: 'R', 'a': 'Go'}
```

deleting items from a dictionary

In [102]:
```
print(d2)

d2.pop(5)
print(d2)

del d2['a']
print(d2)
```
```
{1: 'Python', 2: 'C++', 5: 'R', 'a': 'Go'}
{1: 'Python', 2: 'C++', 'a': 'Go'}
{1: 'Python', 2: 'C++'}
```

membership in a dictionary

In [103]:
```
d3 = dict([ ('Java', ['CS520', 'CS526']), ('Python', ['CS521', 'CS677', 'CS76
7'])])
print(d3)

print('Java' in d3)
print('R' not in d3)
print('C++' in d3)
```
```
{'Java': ['CS520', 'CS526'], 'Python': ['CS521', 'CS677', 'CS767']}
True
True
False
```

Other dictionary methods

keys

In [104]:
```
print(d3)
print(d3.keys())
```

```
{'Java': ['CS520', 'CS526'], 'Python': ['CS521', 'CS677', 'CS767']}
dict_keys(['Java', 'Python'])
```

In [105]:
```
for item in d3:
    print(item)
```

```
Java
Python
```

In [106]:
```
for item in d3.keys():
    print(item)
```

```
Java
Python
```

In [107]:
```
for index, item in enumerate(d3):
    print(index, item)
```

```
0 Java
1 Python
```

values

In [108]:
```
print(d3.values())
```

```
dict_values([['CS520', 'CS526'], ['CS521', 'CS677', 'CS767']])
```

In [109]:
```
for item in d3.values():
    print(item)
```

```
['CS520', 'CS526']
['CS521', 'CS677', 'CS767']
```

items

In [110]:
```
print(d3.items())
```
```
dict_items([('Java', ['CS520', 'CS526']), ('Python', ['CS521', 'CS677', 'CS76
7'])])
```

In [111]:
```
for key, value in d3.items():
    print('{:>8} --> {}'.format(key, value))
```
```
    Java --> ['CS520', 'CS526']
  Python --> ['CS521', 'CS677', 'CS767']
```

Dictionary comprehension

In [112]:
```
powers = {i: 2**i for i in range(10)}
print(powers)
print(powers[8])
```
```
{0: 1, 1: 2, 2: 4, 3: 8, 4: 16, 5: 32, 6: 64, 7: 128, 8: 256, 9: 512}
256
```

In [113]:
```
# Without comprehension

powers = {}
for i in range(10):
    powers[i] = 2**i

print(powers)
```
```
{0: 1, 1: 2, 2: 4, 3: 8, 4: 16, 5: 32, 6: 64, 7: 128, 8: 256, 9: 512}
```

In [114]:
```
# filtering with if

odd_powers = {i: 2**i for i in range(10) if i%2 == 1}
print(odd_powers)
```

```
{1: 2, 3: 8, 5: 32, 7: 128, 9: 512}
```

Sets

- unordered collection of items
- every item is unique
- every item is immutable

In [115]:
```
s1 = set()    # {} is an empty dictionary
s2 = {10, 20, 10, 30}
s3 = set([10, 20, 30, 10, 20])
print(s1)
print(s2)
print(s3)
```

```
set()
{10, 20, 30}
{10, 20, 30}
```

no mutable items in a set

In [116]:
```
s3 = {10, 20, [10, 30]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-116-139e283bd3ac> in <module>
----> 1 s3 = {10, 20, [10, 30]}

TypeError: unhashable type: 'list'
```

no indexing support

```
In [117]: print(s2)
          s2[0]
```

```
{10, 20, 30}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-117-68ce638a5d92> in <module>
      1 print(s2)
----> 2 s2[0]

TypeError: 'set' object does not support indexing
```

Adding elements to a set

```
In [118]: s2 = {10, 20, 10, 30}
          print(s2)

          s2.add(40)
          print(s2)

          s2.add((40, 45))
          print(s2)
```

```
{10, 20, 30}
{40, 10, 20, 30}
{(40, 45), 40, 10, 20, 30}
```

In [119]:
```
s2 = {10, 20, 10, 30}
print(s2)

s2.update([20,30,40,50])
print(s2)
```

```
{10, 20, 30}
{40, 10, 50, 20, 30}
```

In [120]:
```
s2 = {10, 20, 10, 30}
print(s2)

s2.update({10, 15, 25}, [20,30,40,50])
print(s2)
```

```
{10, 20, 30}
{50, 20, 40, 25, 10, 30, 15}
```

Removing items from a set

In [121]:
```
s2 = {10, 20, 10, 30}
print(s2)

s2.remove(10)
print(s2)
```

```
{10, 20, 30}
{20, 30}
```

Membership in a set

In [122]:
```
s2 = {10, 20, 10, 30}
print(s2)

print(10 in s2)
print(40 in s2)
print(50 not in s2)
```

```
{10, 20, 30}
True
False
True
```

Set Union

In [123]:
```
a = {10, 20, 30, 40}
b = {30, 40, 50}

# Union

print(a | b)

print(a.union(b))

print(b.union(a))
```

```
{50, 20, 40, 10, 30}
{50, 20, 40, 10, 30}
{50, 20, 40, 10, 30}
```

Set Intersection

In [124]:
```python
a = {10, 20, 30, 40}
b = {30, 40, 50}

# Intersection

print(a & b)

print(a.intersection(b))

print(b.intersection(a))
```

```
{40, 30}
{40, 30}
{40, 30}
```

Set Difference

In [125]:
```python
a = {10, 20, 30, 40}
b = {30, 40, 50}

# Difference

print(a - b)
print(a.difference(b))

print(b - a)
print(b.difference(a))
```

```
{10, 20}
{10, 20}
{50}
{50}
```

Symmetric difference (except those that are common in both)

In [126]:
```
a = {10, 20, 30, 40}
b = {30, 40, 50}

# Symmetric Difference

print(a ^ b)
print(a.symmetric_difference(b))
```

```
{10, 50, 20}
{10, 50, 20}
```

Set Comparison

In [127]:
```
a = {10, 20, 30, 40}
b = {30, 40, 50}

print(a.isdisjoint(b))
```

```
False
```

In [128]:
```
a = {10, 20, 30, 40}
b = {50, 60}

print(a.isdisjoint(b))
```

```
True
```

```
In [129]: a = {10, 20}
          b = {10, 15, 20, 25}

          print(a.issubset(b))
```

```
True
```

Iterating over sets

```
In [130]: a = {10, 20, 30, 40}
          b = {30, 40, 50}
          c = a | b
          print(c)

          for item in c:
              print (item)
```

```
{50, 20, 40, 10, 30}
50
20
40
10
30
```

```
In [131]: for item in enumerate(c):
              print (item)
```

```
(0, 50)
(1, 20)
(2, 40)
(3, 10)
(4, 30)
```

Functions ... continued

Functions with default arguments

In [132]:
```
def greet(name, className = 'CS677'):
    return ("Hello " + name + ", Welcome to " + className)
```

In [133]:
```
print(greet('John'))
```

```
Hello John, Welcome to CS677
```

In [134]:
```
print(greet('John', 'CS777'))
```

```
Hello John, Welcome to CS777
```

In [135]:
```
def greet(name, className = 'CS677', instructor):
    return ("Hello " + name + ", Welcome to " + className)
```

```
  File "<ipython-input-135-ef53871c95a3>", line 1
    def greet(name, className = 'CS677', instructor):
             ^
SyntaxError: non-default argument follows default argument
```

In [136]:
```
def greet(name, className = 'CS677', instructor = 'Suresh'):
    return ("Hello " + name + ", Welcome to " + className + " - " + instructo
r)
```

In [137]:
```
print(greet('John'))
```

```
Hello John, Welcome to CS677 - Suresh
```

In [138]:
```
print(greet('John', 'CS546'))
```

```
Hello John, Welcome to CS546 - Suresh
```

```
In [139]: print(greet('John', 'CS546', 'Anatoly' ))
```

```
Hello John, Welcome to CS546 - Anatoly
```

```
In [140]: print(greet('John', instructor = 'Anatoly' ))
```

```
Hello John, Welcome to CS677 - Anatoly
```

Functions with variable arguments

```
In [141]: def my_sum(name, *args):
              print(name, args)
              result = 0
              for n in args:
                  result += n
              return (result)
```

```
In [142]: my_sum('John', 1, 2, 3)
```

```
John (1, 2, 3)
```

```
Out[142]: 6
```

Functions with keyword arguments

In [143]:
```
def my_average(name, **kwargs):
    print(name, kwargs)
    result = 0

    if (len(kwargs) == 0):
        return result

    for key, value in kwargs.items():
        print('Score in {} = {}'.format(key, value))
        result += value

    return (result / len(kwargs))
```

In [144]:
```
my_average('John', cs1=70, cs2=80, cs3=90)
```

```
John {'cs1': 70, 'cs2': 80, 'cs3': 90}
Score in cs1 = 70
Score in cs2 = 80
Score in cs3 = 90
```

Out[144]:
```
80.0
```

In [145]:
```
my_average('abc')
```

```
abc {}
```

Out[145]:
```
0
```

Anonymous (Lambda) Functions

In [146]:
```
double = lambda x: 2 * x
```

```
In [147]: print(double(5))
```

```
10
```

```
In [148]: print(double([10,20]))
```

```
[10, 20, 10, 20]
```

```
In [149]: # generally used as argument for higher order functions

          data = [10, 13, 15, 18, 20, 23, 26]

          list(filter(lambda x: (x %2 == 1), data))
```

```
Out[149]: [13, 15, 23]
```

```
In [150]: list(map(double, data))
```

```
Out[150]: [20, 26, 30, 36, 40, 46, 52]
```

```
In [151]: list(map(lambda x: x * x, data))
```

```
Out[151]: [100, 169, 225, 324, 400, 529, 676]
```

Indirect references

In [152]:
```
a = (10, 20, [30,40], 50)
print(a)

b = a[2]
b[0] = 300

print(b)
print(a)
```

```
(10, 20, [30, 40], 50)
[300, 40]
(10, 20, [300, 40], 50)
```

In [153]:
```
# Caution - list references

lists = [[]] * 3

lists[0].append(3)
lists[1].append(5)
lists[2].append(7)

print(lists)
```

```
[[3, 5, 7], [3, 5, 7], [3, 5, 7]]
```

In [154]:
```
# correct approach

lists = [[] for i in range(3)]

lists[0].append(3)
lists[1].append(5)
lists[2].append(7)

print(lists)
```

```
[[3], [5], [7]]
```

In [ ]:
```

```