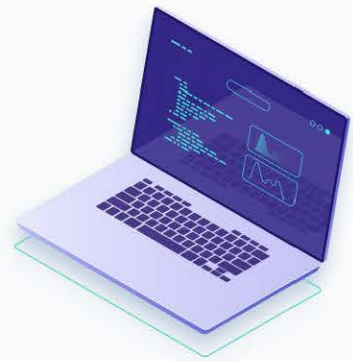
NUS AI SUMMER EXPERIENCE 2019

LISTS, TUPLES & DICTIONARIES



PAN BINBIN

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

OUTLINE

01 LISTS

02 TUPLES

03 DICTIONARIES

01 LISTS

NATURE OF LISTS

```
thingsInPocket = ['phone','keys','wallets','id','pen',5 ]
len(thingsInPocket)
```

```
6
```

```
type(thingsInPocket)
```

```
list
```

```
print(thingsInPocket[0],type(thingsInPocket[0]))
print(thingsInPocket[5],type(thingsInPocket[5]))
```

```
phone <class 'str'>
5 <class 'int'>
```

- Collection of elements grouped together
- Defined by [ ]
- Arbitrary sequence of 6 items in this case
- Datatype

- Extract elements within list by indexing

- Allowed to have different datatypes within a list

METHODS FOR OPERATING ON LISTS

list.append(x)	add item (one or more) to end of list
list.extend(x)	extend list by appending all items from iterable (object you can get item as a sequence one after another)
list.insert(i,x)	insert item at a given position
list.remove(x)	remove the first item from list whose value is x. An error given if no such value
list.pop([j])	remove item from position j and return it. If no index specified, list.pop() removes and returns the last item
list.clear()	removes all items from the list
list.index(x[,start[,end]])	returns the index where the first x is found in the list. start and end limits the search to particular subsequence and is optional.
list.count(x)	return number of times x appears on the list
list.sort(key=None, reverse=False)	sort list in place. allowed to have parameters key and reverse but if ignored, their default values are used.
list.reverse()	reverse elements of list in place
list.copy()	returns a copy of the list

APPENDING THINGS TO LIST

```
thingsInPocket.append('phone')
thingsInPocket
```

```
['phone', 'keys', 'wallets', 'id', 'pen', 5, 'phone']
```

- list.append() method
- 'phone' added to list

```
thingsInPocket.append('coins','chopsticks')
thingsInPocket
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-9e4113416875> in <module>()
----> 1 thingsInPocket.append('coins','chopsticks')
      2 thingsInPocket

TypeError: append() takes exactly one argument (2 given)
```

- more than one argument passed to .append()
- error

```
thingsInPocket.append(['coins','chopsticks'])
thingsInPocket
```

```
['phone', 'keys', 'wallets', 'id', 'pen', 5, 'phone', ['coins', 'chopsticks']]
```

- fix error by passing items as ONE list
- outcome may not be what we want

USING EXTEND TO LIST

```
thingsInPocket.pop()
```

```
['coins', 'chopsticks']
```

- list.pop() method to remove last item on list.
- The output shows the item removed

```
thingsInPocket.extend(['coins','chopsticks'])
thingsInPocket
```

```
['phone', 'keys', 'wallets', 'id', 'pen', 5, 'phone', 'coins', 'chopsticks']
```

- list.extend() method takes 'coin', adds to list, then takes 'chopsticks' to adds to list.
- The extended list is what we want

Sorted function

```
primeNumbers = [11,1,19,2,5,7,13,3,17]
sorted(primeNumbers)
```

```
[1, 2, 3, 5, 7, 11, 13, 17, 19]
```

- sorted() function used to sort the items in primeNumbers

```
primeNumbers
```

```
[11, 1, 19, 2, 5, 7, 13, 3, 17]
```

- the function returns the sorted list

- Look at the two lists. Why are their order different? Did we not sort the list primeNumbers? Why did it remain in the original order?

sorted() function vs .sort() method

```
primeNumbers
```

```
[11, 1, 19, 2, 5, 7, 13, 3, 17]
```

- list.sort method used to sort the items in primeNumbers

```
primeNumbers.sort()
primeNumbers
```

```
[1, 2, 3, 5, 7, 11, 13, 17, 19]
```

- Compare primeNumbers before and after the .sort() method.
- primeNumbers contain the sorted list after sorting

Returning a Value (sorted() function) and in place (.sort() method)

- sorted() sorts the list and returns a sorted list. The original list remains unchanged. The sorted list returned can be stored in another variable.
- .sort() method sorts the list and puts the sorted list in the original list. So the original list is modified. This is what we mean by in place. This is also true for a number of other functions.
- The .sort() method does not return anything. So it is not possible to store the results directly in another variable

- store output of sorted() function in a variable named primeNumbersbySorted

- store output of .sort() method in a variable named primeNumbersbySort

```
primeNumbersbySorted=sorted(primeNumbers)
primeNumbersSortedbySort=primeNumbers.sort()
print('primeNumbersbySorted: ', primeNumbersbySorted, '\n',
      'primeNumbersSortedbySort: ', primeNumbersSortedbySort)
```

```
primeNumbersbySorted:  [1, 2, 3, 5, 7, 11, 13, 17, 19]
 primeNumbersSortedbySort:  None
```

- primeNumbersbySorted contains the sorted list

- primeNumbersbySort contains nothing

RANGE CONSTRUCTION

- range(n) generates integers from 0,1,2 . . . , n-1
- range(start,stop,step) generates a sequence of integers. The sequence is

start, start+step, start+2*step and so on, up to but not including stop.

If step is omitted, the default is 1.

- range(5) contains (0,1,2,3,4)
- range(0,10,3) contains (0,3,6,9)
- range(4,7) contains (4,5,6)

USE RANGE TO CONSTRUCT A LIST

- Suppose you wish to construct a list containing temperatures in Celcius
- [-5, 0 , 5, 10, 15, 20, 25, 30, 35, 40]
- You could type in the numbers yourself, but if there are many, it would be too tedious.
- So we use the range function together with the for loop

```
Cdegrees=[ ]                    #define an empty list you going to fill later
for i in range(-5,41,5):        #carry out operations in the loop starting with i=5,
                                #increase by 5 for each loop, and stop at the largest
                                #number before 41
   Cdegrees.append(i)           #append each i to the list called Cdegrees, with each loop
Cdegrees                        #check what is in the list
```


LIST COMPREHENSION

- Efficient method to construct list
- newlist = [E(e) for e in list]
- E(e) represents an expression involving element e
- Convert the list of temperature in Cdegrees from Celcius to Farenheit and store the result in a list called Fdegrees
- Fdegrees = [(9/5)*c + 32 for c in Cdegrees]

```
Fdegrees =[(9/5)*c+32 for c in Cdegrees]
Fdegrees
```

```
[23.0, 32.0, 41.0, 50.0, 59.0, 68.0, 77.0, 86.0, 95.0, 104.0]
```

EXERCISE

USE LIST COMPREHENSION FOR THE FOLLOWING

- 1. CREATE A MULTIPLICATION TABLE FOR MULTIPLYING 13 FROM 1 TO 12
- 2. CREATE A LIST SHOWING THE CONVERSION RATE OF 1 TO 100 SGD TO USD AT RATE OF USD1 = SGD1.34
- 3a. Create a list, called numberlist, containing integers from 1 to 100
- 3b. FROM numberlist, EXTRCT ALL THE NUMBERS THAT CAN BE DIVIDED BY 12 AND PUT IT IN A NEW LIST

ZIP FUNCTION TO COMBINE TWO LISTS


```
result=zip(Cdegrees, Fdegrees)
result
```

(-5,23)
(0, 32)
(5, 41)
. . .

- tuples created.
- n^{th} element from Cdegrees and n^{th} element from Fdegrees becomes n^{th} tuple in result
- iterator which are lazily evaluated.

```
Out[40]: <zip at 0x3e34f9a788>
```

```
In [41]: print(list(result))
         [(-5, 23.0), (0, 32.0), (5, 41.0), (10, 50.0), (15, 59.0), (20, 68.0), (25, 77.0), (30, 86.0), (35, 95.0), (40, 104.0)]
In [35]: print(list(result))
         []
```

- convert result to list using a list(result) and printing out
- Note that the iterator result can only be evaluated once
- The iterator is exhausted after evaluation and is now empty

PRINTING TWO LISTS USING ZIP FUNCTION

At each iteration, assign:
first element of each tuple to C
second element of each tuple to F

produces iterator with tuples containing elements from Cdegrees and Fdegrees

```
for C,F in zip(Cdegrees,Fdegrees):
    print(C,F)
```

```
-5 23.0
0 32.0
5 41.0
10 50.0
15 59.0
20 68.0
25 77.0
30 86.0
35 95.0
40 104.0
```


USEFULNESS OF ZIP FUNCTION

- MATRIX OPERATION
- CREATE DICTIONARY
- COMBINING IOT DATA
  - SET OF CLIMATE DATA COLLECTED OVER 5 MINUTES INTERVAL
  - COLLECTS DATA FOR TEMPERATURE, HUMIDITY, WINDSPEED
  - USE 3 IOT DEVICE
    - TEMPERATURE =[28, 28.7, 28.5 ....]
    - HUMIDITY = (70, 72, 75 ...]
    - WINDSPEED = [2, 2.2, 2.1 ...]
    - COMBINE THE TEMPERATURE, HUMIDITY, WINDSPEED DATA THAT BELONG TOGETHER
    - READINGS = [(28,70,2), (27.7,72,2.2), (28.5,75,2.1)...]

02 TUPLES

NATURE OF TUPLES

- SIMILAR TO LISTS
- DIFFERENCE: IMMUTABLE (CANNOT BE CHANGED)
- WHY TUPLES INSTEAD OF LIST
  - PROTECT AGAINST ACCIDENTAL CHANGE IN CONTENT
  - CODE USING TUPLES FASTER THAN USING LISTS

TUPLE REPRESENTATION

tuple1 = (1,2,3,4,5,'numbers')
tuple2 = 10,'cool', True

- Tuple defined by either ( ); or No Brackets at all

- Elements within tuple accessed by their index

```
print(type(tuple2[0]),type(tuple2[1]),type(tuple2[2]))
```

```
<class 'int'> <class 'str'> <class 'bool'>
```

- Tuple can contain elements of different data types

```
tuple2[0]=20
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-ad9ee80c99a1> in <module>()
----> 1 tuple2[0]=20

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

- Cannot change value of element in tuple (immutable)

What do you notice about the difference between the two?

```
type((20))
```

```
int
```

```
type((20,40))
```

```
tuple
```

03 DICTIONARIES

WHAT IS A DICTIONARY?

- List – collection of elements indexed from 0 to n-1
- Dictionary – the elements (called values here) are indexed by text instead of position

DICTIONARY ILLUSTRATION AND SYNTAX

Suppose you have a list of temperatures of the following cities:
Amsterdam, Hong Kong, London, San Francisco, Singapore.

temps = [17, 26, 9, 20, 34]
#List shows the temperatures of the cities above in order

What is the temperature of Singapore?

```
temps[4]  #use index to extract temperature of Singapore
```

```
34
```

Must remember the index position to recall the temperature.

- Dictionary enclosed by curly brackets
- text-value pairs are created

```
cityTemps = {'Amsterdam':17, 'Hong Kong':26, 'London':9,
             'San Francisco':20, 'Singapore':34}
```

```
cityTemps['Singapore']
```

```
34
```

- Text is called the key
- Easier to call up the temperature of Singapore

ALTERNATIVE METHODS OF DEFINING DICTIONARY

```
cityTempsEqual = dict(Amsterdam=17, HongKong=26, London=9,
                SanFrancisco=20, Singapore=34)
cityTempsEqual['HongKong']
```

```
26
```

- Using the dict function
- Assigning value to text using "="

```
temps = [17, 26, 9, 20, 34]
```

```
city = ['Amsterdam', 'Hong Kong', 'London', 'San Francisco', 'Singapore']
cityTempsZip = dict(zip(city,temps))
cityTempsZip['San Francisco']
```

```
20
```

- Using the dict function
- Applying the zip function to two lists
- List with name of city is first (key)

COMMON DICTIONARY OPERATIONS

- ADDING TO DICTIONARY

Define new key in existing dictionary

Assign value to the new key

```
cityTemps['Sydney']=23
cityTemps
```

```
{'Amsterdam': 17,
 'Hong Kong': 26,
 'London': 9,
 'San Francisco': 20,
 'Singapore': 34,
 'Sydney': 23}
```

- DELETING ELEMENTS FROM DICTIONARY

Using the del function

key of element to be deleted

```
del cityTemps['Sydney']
cityTemps
```

```
{'Amsterdam': 17,
 'Hong Kong': 26,
 'London': 9,
 'San Francisco': 20,
 'Singapore': 34}
```

The key together with its value is deleted

COMMON DICTIONARY OPERATIONS

- LOOK UP VALUES OF KEYS IN DICTIONARY

```
cityTemps['Singapore']
```

```
34
```

- Type in key
- Get result

```
cityTemps['Berlin']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-46-63278bcb9c0e> in <module>()
----> 1 cityTemps['Berlin']

KeyError: 'Berlin'
```

- Get error when key not in dictionary
- Problematic when you are processing data and the program will gets stopped by the error

DEFAULT VALUES IN DICTIONARY

Use the .get() method with two arguments

First argument is the key

Second argument is the default value

```
cityTemps.get('Singapore','Temperature not listed')
```

```
34
```

If key is found in dictionary, the value from the key-value pair is returned

```
cityTemps.get('Berlin','Temperature not listed')
```

```
'Temperature not listed'
```

If key is not found in dictionary, the default value is returned

DISORDERLY BEHAVIOUR OF DICTIONARIES

- Number of clients in last 6 months: Jan-Jun
- Enter data in order of months (Jan first, followed by Feb and so on) in dictionary
- You would think that it would print out in the same order that you input it in?
- Surprise! It does not

```
clientNumber = {'Jan': 10, 'Feb': 9, 'Mar': 14,
                'Apr': 12, 'May': 5, 'Jun': 6}
list(clientNumber.items())
```

```
[('Mar', 14), ('Jun', 6), ('Jan', 10), ('Apr', 12), ('Feb', 9), ('May', 5)]
```

- Output of dictionary not in the order you input it

ORDERING YOUR DICTIONARIES

- import the OrderedDict container from the collections module
- It provides this special container as alternative to the general purpose dict function

```
from collections import OrderedDict
clientNumberOrdered = OrderedDict([('Jan', 10), ('Feb', 9), ('Mar', 14),
                                   ('Apr', 12), ('May', 5), ('Jun', 6)])
list(clientNumberOrdered.items())
```

```
[('Jan', 10), ('Feb', 9), ('Mar', 14), ('Apr', 12), ('May', 5), ('Jun', 6)]
```

- the dictionary is now printed out in the order that it was entered
- this is useful when the data you read in from your database is ordered.

DEFINING SPARSE MATRIX WITH DICTIONARIES (1)

$$\begin{vmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{vmatrix}$$

- row 1, column 4
- with 0 indexing (first row/column starting with 0)
- (row,column) = (0,3)
- (0,3) has value of 1

- Mostly zeros as elements
- Inefficient to represent as a list with all elements
- Use dictionary to represent with row and column index as key

DEFINING SPARSE MATRIX WITH DICTIONARIES (2)

matrix = {(0,3):1 , (2,1):2 , (4,3):3}

- matrix defined as a dictionary
- only 3 key-value pairs
- those with non-zero value

```
print(matrix.get((0,3),0),'\n',
matrix.get((0,0),0))
```

```
1
 0
```

- use .get() method to access the matrix
- returns the value if key is in dictionary (non-zero element
- returns 0 if key is not in dictionary (zero elements)

DICTIONARIES AS COUNTER

- Dictionaries provide elegant way to count number of occurrences of letter or digits in a string
- The key in the key-value pair is the letter (or digit) you want to count
- The value in the key-value pair is the number of occurrence of the letter (of digit)
- Does not have to store letters or digits that are not present in the string

COUNTING OCCURRENCE OF LETTER

- Take the string 'Mississippi'
- Count occurrence of each letter that is in the string

- Define a dictionary called letter_counts
- Initially empty

- Iterates through by taking one letter at a time

```
letter_counts = {}
for letter in 'Mississippi':
    letter_counts[letter]=letter_counts.get(letter,0)+1
letter_counts
```

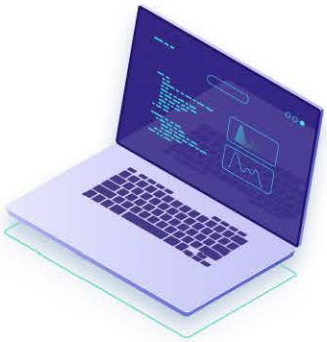
- Add the key (if not already present) to the dictionary, one at a time

```
{'M': 1, 'i': 4, 'p': 2, 's': 4}
```

- Output of the program

- First time key is added, it is given a value of 1

- Subsequent an existing key is found, a value of 1 is added to its existing value



PAN BINBIN

BINBIN.PAN@NUS.EDU.SG

NUS AI SUMMER EXPERIENCE 2019

LISTS, TUPLES & DICTIONARIES