Introduction to Python: Lists and Tuples

Creating a variable which can store multiple values

Note that we are using "***Python 3***"

Created by John C.S. Lui, May 24, 2018.

Important note: *If you want to use and modify this notebook file, please acknowledge the author.*

Lists

A collection of items which can be stored in a *list* variable

In [ ]:
```
professors = ['james', 'john', 'patrick', 'michael fung', 'siu hang']

for teacher in professors:
    print ('Hello, ' + teacher.title() + ', you are so butt ugly !!!')
```

Defining a list

In Python, square brackets define a list. To create a list, provide a name for the list, add the equals sign, and the values you want within square brackets

In [ ]:
```
cse_teachers = ['John C.S. Lui', 'Patrick P.C. Lee', 'James Cheng', 'W
print (cse_teachers)
print ("first element in the list is =", cse_teachers[0])
print ("last element in the list is =", cse_teachers[-1])   # useful i
```

Accessing various elements in a list

In [ ]:
```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',
print ("The second element is =", cse_teachers[1].title())   # see how
print ("The second to the last element is =", cse_teachers[-2].title()
```

In [ ]:
```
# we can't use a number larger (or less than) the dimension of the lis
print ("first element in the list is =", cse_teachers[6])
#print ("first element in the list is =", cse_teachers[-7])
```

List operations

So, what are some of the built-in functions for list? Let's check.

In [ ]:
```
my_list = [1,2,3]
dir(my_list)
```

In [ ]:
```
help(my_list.append)
```

Exercise

Create a list with various programming languages that you know, then print them out

- in increasing order of the index
- in decreasing order of the index

List looping

A loop is a block of code that repeats itself until it uses up all items in the list (or until a condition is met). Let's try looping which runs once for every item in thelist.

In [ ]:
```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',

index = 0
for teacher in cse_teachers :
    print("for index =", index, "the teacher is", teacher.title())
    index = index + 1
```

In [ ]:
```
# pay attention to indentation
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',

index = 0
for teacher in cse_teachers :
    if index < 3:
        print ("For teacher", teacher.title(), ", I really like this d
    else:
        print("For teacher", teacher.title(), ", I really hate this du
    index = index + 1

print("So why all teachers in the first floor look like dorks?")
```

Enumerating a list

In [ ]:
```
'''
If you want to know the 'index' of the item, you can use either
  1. list.index(value)
  2. enumerate() function tracks the index of each item
'''
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',

print("Teachers in the 1st floor are:\n")

print ("------- use enumerate -----------")
for index, teacher in enumerate(cse_teachers):
    position = str(index)        # change index to string so we can us
    print("Position: " + position + " Teacher: " + teacher.title())

print ("------- use list.index(value) -----------")
for teacher in cse_teachers:
    print("Position:", cse_teachers.index(teacher), "Teacher: " + teacl
```

List operations

Modifying an item in the list

In [ ]:
```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',

cse_teachers[0] = 'prof. john c. s. lui'    # we are changing the firs
print(cse_teachers[0].title())
print(cse_teachers)
```

Locating an item in the list

In [ ]:
```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',

print("index of 's. h. Or' is: ", cse_teachers.index('s. h. Or'))
```

In [ ]:
```
# check what this returns
print (cse_teachers.index('michael lyu'))    # It will output ValueErr
```

Testing for item membership

In [ ]:
```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',

print ("Is 'john c. s. lui' in the list?", 'john c. s. lui' in cse_tea
print ("Is 'John C. S. Lui' in the list?", 'John C. S. Lui' in cse_tea
```

Adding item into a list

Appending to the end of the list

In [ ]:
```
# use the 'append' method

cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',
cse_teachers.append('yat chiu law')   # append to the end of the lsit

for teacher in cse_teachers:
    print (teacher.title(), "is so cool !!!!")
```

Inserting an item to a list

In [ ]:
```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng',

print (cse_teachers)
cse_teachers.insert(2,'james bond')  # inserting into position 2

print (cse_teachers)

del cse_teachers[2]                  # removing an item

print (cse_teachers)
```

Creating an empty list

In [ ]:
```
# Create an empy list
Turing_winners = []
print("The list of Turing_winners is = ", Turing_winners)

# Add some items in the list
Turing_winners.append('donald knuth')     # appending to the end
Turing_winners.append('vint cerf')
Turing_winners.append('andrew yao')
Turing_winners.append('leslie valiant')

print("The list of Turing_winners is = ", Turing_winners,"\n")

for winner in Turing_winners:
    print ("The work of", winner.title(), "is fantastic !!!")
```

Sorting items in a list

In [ ]:
```
# sort items in alphabetical order
Turing_winners.sort()   # we sort and made update to the list
print ("The sorted Turing_winners list is", Turing_winners)
```

In [ ]:
```
# Let's sort it in reverse order
Turing_winners.sort(reverse=True)    # use option to sort in reverse o
print ("The sorted Turing_winners list is", Turing_winners)
```

Difference between sort() and sorted()

When using *sort()*, you can't recover the original order. If you just want to display a list in sorted order, but preserve the original order, you can use the *sorted()* function.

In [ ]:
```
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie v

print ("Turing_winners is", Turing_winners)

# Display in sorted fashion but maintain the original ordering

Turing_winners_sorted = sorted(Turing_winners)  # use of sorted() func

print("Turing_winner is", Turing_winners)
print("Sorted version is", Turing_winners_sorted)

# Display in reverse sorted fashion
print("\nPring the original list in reverse sorted fashion:")
for winner in sorted(Turing_winners, reverse=True):
    print("\t", winner.title())
```

Reverse a list from its original order

We can use the list built-in function list.reverse()

In [ ]:
```
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie v

print ("Original ordering:", Turing_winners)

Turing_winners.reverse()     # using the built-in function list.revers

print ("Reverse ordering:", Turing_winners)
```

Sorting a numerical list

All the sorting functions we metioned applied for a numerical list

In [ ]:
```
# Define a numerical list
numbers_list = [1, 7, 3, 5, 13, 11]

numbers_list.sort()          # sort in increasing order
print ("The sorted list of numbers_list:", numbers_list)

numbers_list.sort(reverse=True)   # sort in decreasing order
print ("The reversed sorted list of numbers_list:", numbers_list)

print(numbers_list)    # see the permanent change?
```

In [ ]:
```
# Define a numerical list
numbers_list = [1, 7, 3, 5, 13, 11]

# sorted() preserve the original order and return a list !!!
print ("The sorted list of numbers_list:", sorted(numbers_list))
print ("The reverse sorted list of numbers_list:", sorted(numbers_list

print(numbers_list)
```

Determining the length of a list

This can be achieved via the len() function

In [ ]:
```
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va
winners_count = len(Turing_winners)   # len() return an integer

print("The length of the Turing_winners list is:", winners_count)
```

In [ ]:
```
## We can keep adding elements to the list and still find the length
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va

Turing_winners.append('michael stonebraker')
Turing_winners.append('barbara liskov')

winners_count = len(Turing_winners)

print("The length of the Turing_winners list is:", winners_count)
print("The length of the Turing_winners list is: " + str(len(Turing_wi
```

Removing some items from a list

We remove items from a list through (1) their position, or (2) through their value.

In [ ]:
```
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va

# remove and insert an item through position via del() and the list bu
del Turing_winners[1]     # delete via position
print(Turing_winners)
Turing_winners.insert(1,'vint cerf')     # insert via position
print (Turing_winners)
```

In [ ]:
```
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va

# remove an item via its value via the list.remove('arg')
Turing_winners.remove('vint cerf')
print(Turing_winners)
Turing_winners.insert(1,'vint cerf')   # insert it back
print(Turing_winners)
```

In [ ]:
```
# Note that the built-in function only removes the "first occurance" o
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va

Turing_winners.append('vint cerf')
print (Turing_winners)

Turing_winners.remove('vint cerf')     # remove the FIRST OCCURANCE of
print (Turing_winners)
```

Heterogenous items in a list

In [ ]:
```
# list with different types of items
my_list = ['john c. s. lui', 100, 10.01, 'jack', ['Bill Gates', 30, 'S

for item in my_list:
    print('item is:', item)
```

The beauty of popping items from a list

The function *pop()* removes the last item from the list and returns the item to user

In [ ]:
```
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va

Turing_winners.append('barbara liskov')
last_item = Turing_winners.pop()    # popping the last item and assign
print ("The popped item is", last_item.title())
print ("The Turing_winners is:", Turing_winners)
```

In [ ]:
```
# One can pop any item he/she wants from a list, by giving the index o
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va
Turing_winners.append('barbara liskov')
item = Turing_winners.pop(0)   # popping the first item
print(Turing_winners)
print("Items that was popped is", item)

Turing_winners.pop(0)
print (Turing_winners)
Turing_winners.insert(0, 'vint cerf')
Turing_winners.insert(0, 'donald knuth')
print (Turing_winners)
```

Slicing operations

If you want to get a subset of items from a list, slicing is a convenient way to achieve it

In [ ]:
```
# Obtain the first 4 items
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va
Top_four_names = Turing_winners[0:4]  # first index shows the begin ind

for winner in Top_four_names:
    print('The first four winners:', winner.title())
```

Grap from the beginning to a particular stopping index

In [ ]:
```
# Obtain the first 3 items
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va
Top_three_names = Turing_winners[:3]   # start from index 0 and stop i

for winner in Top_three_names:
    print("First three names are:", winner.title())
```

In [ ]:
```
# A cool way to grap EVERYTHING without knowing the dimension
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va
Top_names = Turing_winners[:-1]   # start from index 0 and stop in ind

for winner in Top_names:
    print("Winners are:", winner.title())
```

In [ ]:
```
# This should be obvious
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va

Names = Turing_winners[1:3]  # get two (3-1) items, starting from inde
print(Names)

Names = Turing_winners[2:]  # get all items starting from index 2
print(Names)
```

Duplicating a list and manipulate it

In [ ]:
```
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie v

T_winners = Turing_winners[:]

# let's remove the first three items from the duplicated list

del T_winners[0]
del T_winners[0]
del T_winners[0]
print ("T_winners:", T_winners)
print ("Turing_winners is not changed:", Turing_winners)
```

Exercise

- Create a list with the first 20 integers, starting from 0
- print the first 7 items out
- remove the first 5 items
- remove the last 3 items
- print out 4 items, starting from index 2

Numerical list and related functions

Introduce some functions which make working with numerical lists more efficient.

In [ ]:
```
# Create a numerical list with the first 20 numbers
numbers_list = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

for number in numbers_list:
    print("The number is:", number)

#   IT WILL BE A PAIN TO CREATE A NUMERICAL LIST WITH THE FIRST MILLIO
```

The *range()* function

range(int1, int2) where int1 is the beginning integer and int2 is the stopping integer

In [ ]:
```
# Let's repeat our previous code
for number in range(1,20):
    print("The number is:", number)
```

In [ ]:
```
# Let's try range(int1,int2,inc) where int1 is the beginning integer,
for number in range(1,20,2): # Generate only odd numbers
    print("The odd numbers are:", number)
```

In [ ]:
```
# Let's try to store the numbers we generate from range() to a list
numbers_list = list(range(1,20,2))
print ("The new numbers_list is:", numbers_list)
```

In [ ]:
```
# Store the first 2 million numbers in a list.
numbers_list = list(range(1,2000001))

# Show the length of the list:
print("The numbers_list has " + str(len(numbers_list)) + " numbers in

# Show the last five numbers:
print("\nThe last five numbers in the list are:")
for number in numbers_list[-5:] :
    print(number)
```

Finding the minimum, maximum and aggregate of a numerical list

We can use functions like *min(), max(), sum()*

In [ ]:
```
grades = [98.0, 20.0, 15.5, 88.0, 60.0, 55.5, 89.0, 95.0, 82.2, 55]  #

minimum_grade = min(grades)  # find the minimum in the list
maximum_grade = max(grades)  # find the maximum in the list
average_grade = sum(grades)/len(grades)    # find the average grade

print("The lowest grade is:", minimum_grade)
print("The highest grade is:", maximum_grade)
print("The average grade is:", average_grade)
```

Exercise

- Generate a numerical list with the first 100 integers, starting from 1
- Calculate the sum (you should know this via the geometric series)
- Generate another numerical list with the first 100 even numbers
- Calculate the minimum, maximum, sum and average of the second numerical list

List comprehension

It is a shorthand approach to create and manipulating a list. It is very common in Python programming

In [ ]:
```
# Let's create a numerical list with square root of the first 20 numbe
square_roots = []     # create an empty list

for number in range(1,21):
    square_roots.append(number**0.5)   # generate the square root of t

for number in square_roots:
    print("square root:", number)
```

Let's see how we can use "list comprehension" to write a cleaner code

In [ ]:
```
square_roots = [number**0.5 for number in range(1,21)]  # the power of

for number in square_roots:
    print("square root:", number)
```

Non-numerical list comprehension

Let's create an initial list, and use a list comprehension to create a second list from the first one

In [ ]:
```
# First list
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va

# Second list
second_list = []
for winner in Turing_winners:
    second_list.append("Prof. "+ winner.title() + ", this person is br

for winner in second_list:
    print("Here comes "+ winner)
```

In [ ]:
```
## Let's use list comprehension
Turing_winners = ['donald knuth', 'vint cerf', 'andrew yao', 'leslie va
second_list = ["Prof. "+ winner.title() + ", this person is brillant!!

for winner in second_list:
    print("Here comes "+ winner)
```

Exercise

- Make a list of the first ten multiples of ten (10, 20, 30... 90, 100). There are multiple ways to do it. Try different ways.
- Make a list of the first ten cubes (1, 8, 27... 1000) using a list comprehension.
- Store five professor names in a list. Make a second list that adds the phrase "is aweful !" to each name, using a list comprehension.

String and characters

We often need to grap a string and process each character within it. Let's see how we do it via list

In [ ]:
```
message = "CSCI2040 is really boring !!!"

for char in message:   # print each character out
    print("the character just read in is: ", char)
```

In [ ]:
```
# We can create a list from a string. The list will have one element f
message = "CSCI2040 is really boring !!!"

message_list = list(message)   # list() converts a string to a list, w
print(message_list)
```

In [ ]:
```
# Now we can use various functions and slicing operations to access ea

message = "CSCI2040 is really boring !!!"
message_list = list(message)

print("first character is:", message_list[0], "last charcter is:", mes
print("\nThe first 3 characters are:", message_list[:3])  # Remember s
print("The last 4 characters are:", message_list[-4:])
```

Finding substrings within a string

At times, we may want to find a particular words (or patterns) within a large string (document)

In [ ]:
```
message = "CSCI2040 is really boring !!!"
flag = 'boring' in message    # it returns True or False
print ("The flag is:", flag)
flag = 'exciting' in message
print("This time the flag is:", flag)
```

In [ ]:
```
# The find() method in the string class indicates the index at which t
message = "CSCI2040 is really boring !!!"
flag_index = message.find('boring')     # locate the index
print("For 'boring' sub-string, it appears in index", flag_index, "in
```

In [ ]:
```
# Pay attention that find only locates the FIRST OCCURANCE of the subs
message = "CSCI2040 is really boring !!!, really very boring."
flag_index = message.find('boring')
print('The index is:', flag_index)

# If you to find the LAST OCCURANCE of the substring, use rfind()
flag_index = message.rfind('boring')
print('The index of the last occurrence is:', flag_index)
```

Replacing substrings

Use the *replace()* method to replace any substring with another substring

In [ ]:
```
message = "CSCI2040 is really boring !!!, really very boring."
message = message.replace('boring', 'exciting')
print (message)
```

Couting substring occurences

Use *count()* method to determine how many times a substring appears within a string

In [ ]:
```
message = "CSCI2040 is really boring !!!, really very boring."
number  = message.count('boring')
print("The number of times 'boring' appears in 'message' is:", number)
```

Splitting strings

At times, we find it useful to break a string into, say words.
The *split()* method returns a list of substrings.
The *split()* takes takes one argument, the character that separates the parts of the string.

In [ ]:
```
message = "CSCI2040 is really boring !!!, really very boring."
words = message.split(' ')  # split uses ' ' (or space) as separator
print(words)

words = message.split('really')  # split uses 'really' as separator
print(words)
```

Other methods for the string class

There are many useful methods for the string class, please refer
https://docs.python.org/3.3/library/stdtypes.html#string-methods
(https://docs.python.org/3.3/library/stdtypes.html#string-methods)

Interesting exercise

Now you know string and various methods, you can take on the following exercise.
The exercise is to **Counting DNA Nucleotides** Please refer to
Counting DNA Nucleotides (http://rosalind.info/problems/dna/).

Tuples

Tuples are lists that can **never be changed** (or immutable object).
Tuple uses left/right parenthesis.

In [ ]:
```
some_NBA_teams = ('LA Lakers', 'Golden State Warriors', 'Cleveland Cav

print('The first team in the list is:', some_NBA_teams[0], "\n")  # Us

for team in some_NBA_teams:
    print("NBA team: " + team)
```

In [ ]:
```
# Can we add things to a tuple?
some_NBA_teams.append('Chicago Bulls')

# Can we del items from a tuple ?
# del some_NBA_teams[0]
```

Tuple and List conversion

How can we convert a tuple (list) to a list (tuple)? Let's illustrate.

In [ ]:
```
my_tuple = (1,2,3,4,5)
my_list = list(my_tuple)
print('my_tuple = ', my_tuple)
print('my_list = ', my_list)

my_tuple2 = tuple(my_list)
print("my_tuple2 =", my_tuple2)
print ("The types are: ", type(my_tuple), type(my_list), type(my_tuple
```

Formatted output

Sometimes, we may want to print out something similar to the C or C++ style like %d, %s, ..etc

In [ ]:
```
some_NBA_teams = ('LA Lakers', 'Golden State Warriors', 'Cleveland Cav

index = 1
for team in some_NBA_teams:
    print("The #%d team in NBA is: %s" %(index, team))
    index = index + 1
```

Advice on Python Style

It is important to be a *good* and *elegant* programmer. These are some suggestions

- Use 4 spaces for indentation
- Use **UP** to 79 characters per line of code, and 72 characters for comments.
- Use single blank lines to break up your code into meaningful blocks.
- For comment, use a single space after the pound sign at the beginning of a line.
- Name variables and program files using only lowercase letters, underscores, and numbers.

In [ ]: