ISE 101 – Introduction to Information Systems

- Lecture 4 Objectives:
  - Lists
  - For loops

LISTS

Lists

- List is an array of values
- A list can be formed using squared brackets […]
- The values in list are called ‘elements’ or ‘items’
- Items within the list should be separated using comma
- A list can be
  - an array of integer numbers

    x=[1, 5, -5]
  - an array of floating point numbers

    x=[-3.4, 4.67, 34.56]

Lists

- Contrary to other programming languages, Python lists
  - may contain different types
  - can be extended/shrinked dynamically
- The following list contains a string, a float, an integer, and another list:

  ['spam', 2.0, 5, [10, 20]]
- A list within another list is said to be nested
- A list that contains no elements is called an empty list; you can create one with empty brackets, [].

Indices of Lists

- List indices work the same way as string indices:
  - Any integer expression can be used as an index.
  - If you try to read or write an element that does not exist, you get an error.
  - If an index has a negative value, it counts backward from the end of the list.

Indices of Lists

```
>>> L=[3,4,5.0,6+0j,'test','String']
>>> L[0]
3
>>> L[2]
5.0
>>> L[-1]
'String'
>>> L[23]
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.IndexError: list index out of range
>>> L[-23]
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.IndexError: list index out of range
```

List Slices

- Same as strings slices, sublists of items can be accessed using slices

```
>>> L=[3,4,5.0,6+0j,'test','String']
>>> L[:3]
[3, 4, 5.0]
>>> L[-3:0]
[]
>>> L[-3:-1]
[(6+0j), 'test']
>>> L[-1:-3:-1]
['String', 'test']
>>> L[2::2]
[5.0, 'test']
>>> L[:-4]
[3, 4]
```

“len” with Lists

- “len(list_variable)” returns the number of items within the list
- Positive indices start from 0 to len(list)-1
- Negative indices start from –len(list) to -1

```
>>> L=[3,4,5.0,6+0j,'test','String']
>>> len(L)
6
>>> L[len(L)-1]
'String'
>>> L[-1:-len(L):-1]
['String', 'test', (6+0j), 5.0, 4]
```

“in” with Lists

- “in” operator was used to check the existence of a character or substrings within a string ie.

  >>> ‘ic’ in ‘Nice’

  >>> True

- Similarly, “in” can be used to check the existence of an element in a list

  >>> 3 in [3,4,5]

  >>> True

- True is returned as 3 is an item within the [3,4,5]

“in” with Lists

```
>>> L=[3,4,5.0,6+0j,'test','String']
>>> 5 in L
True
>>> 6 in L
True
>>> 'est' in L
False
>>> 'Test' in L
False
>>> 'test' in L
True
```

“+” and “*” Operators for Lists

- Same as strings
  - “+” is used to merge lists into a single list

```
>>> L1=[2, 4.9]
>>> L2=[3,'etc']
>>> L3=[L1,7]
>>> L1+L2
[2, 4.9, 3, 'etc']
>>> L1+L2+L3
[2, 4.9, 3, 'etc', [2, 4.9], 7]
```

“+” and “*” Operators for Lists

- Same as strings
  - “*” is used to repeat list

```
>>> L1=[2, 4.9]
>>> L2=[3,'etc']
>>> L3=[L1,7]
>>> L1*3
[2, 4.9, 2, 4.9, 2, 4.9]
>>> L1*3+L2+L3*2
[2, 4.9, 2, 4.9, 2, 4.9, 3, 'etc', [2, 4.9], 7,
[2, 4.9], 7]
>>> L1*3.0
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.TypeError: can't multiply sequence by
non-int of type 'float'
```

Lists are Mutable

- What is the difference of S and L

  S="test"

  L=['t','e','s','t']
- S is of type string and L is of type list
- Strings are immutable.
- Once generated, characters within a string cannot be changed
- Lists are mutable
- Items of a list can be changed

Lists are Mutable

```
>>> S='test'
>>> S[1]='c'
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.TypeError: 'str' object does not
support item assignment
>>> L=['t','e','s','t']
>>> L[1]='c'
>>> L
['t', 'c', 's', 't']
```

Conversion from String to List

- It is possible to convert a string to list of characters using “list(string_variable)”

```
>>> S='test'
>>> list(S)
['t', 'e', 's', 't']
```

“range”

- Frequently, lists with adjacent integers are needed
- A special function “range” can be used to generate these integer lists
- list(range(N)) creates a list of integers that start from 0 and goes upto N-1 by increments of 1

```
>>> L=range(4)
>>> L
range(0, 4)
>>> list(L)
[0, 1, 2, 3]
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

“range”

- If two arguments are used in range, integer list includes the first argument, goes upto second argument (excludes the second argument) by increments of 1
- list(range(N1,N2)) will generate a list of integers that start from N1 and goes to N2-1 by increments of 1
- If N1>=N2 an empty list will be returned

```
>>> list(range(3,7))
[3, 4, 5, 6]
>>> list(range(7,3))
[]
>>> list(range(3,3))
[]
```

“range”

- A third argument can be given to “range” to change the step size
- Default step size is 1
- list(range(N1,N2,S)) will generate a list of integers that starts from N1 and goes upto N2-1 with steps of S

```
>>> list(range(3,17,4))
[3, 7, 11, 15]
>>> list(range(7,3,-1))
[7, 6, 5, 4]
>>> list(range(7,3,-2))
[7, 5]
>>> list(range(7,3,-5))
[7]
```

List Extension

- A list may be extended with another list using the “extend” method of a list
- Items of the second list is added to the items of the list
- There are two ways of using “extend” method (this is valid for all methods)
  - list_variable1.extend(list_variable2)
  - list.extend(list_variable1,list_variable2)
- In both ways, items of list_variable2 will be added to END of the items of list_variable1
- Items of list_variable2 will not be effected

List Extension

- What is the difference between “+” operator and “extend” method ?
- “+” operator do not change the items of its arguments, but generate a new list

```
>>> L1=[2, 4.9]
>>> L2=[3, 'etc']
>>> L1+L2
[2, 4.9, 3, 'etc']
>>> L1
[2, 4.9]
>>> L1.extend(L2)
>>> L1
[2, 4.9, 3, 'etc']
```

List Extension

```
>>> L1=[2, 4.9]
>>> L2=[3, 'etc']
>>> list.extend(L1,L2)
>>> L1
[2, 4.9, 3, 'etc']
>>> L2
[3, 'etc']
```

Dynamic Items of List

- New elements (items) can be added to a list using “append” method
  - list_variable.append(new_item)
  - list.append(list_variable,new_item)

```
>>> L1=[2, 4.9]
>>> L1.append("s")
>>> L1
[2, 4.9, 's']
>>> list.append(L1,3+4j)
>>> L1
[2, 4.9, 's', (3+4j)]
```

Pop Items from Lists

- “pop” method deletes the last item of a list and returns its value

```
>>> L1=[2, 4.9]
>>> tmp=L1.pop()
>>> tmp
4.9
>>> L1
[2]
>>> tmp=L1.pop()
>>> tmp
2
>>> L1
[]
```

Pop Items from Lists

- It is possible to “pop” another item other than the last one
- Index of the item is given as an argument to “pop” method

```
>>> L1=[2, 4.9, 5, 7]
>>> tmp=L1.pop(2)
>>> tmp
5
>>> L1
[2, 4.9, 7]
```

Deleting Items from Lists

- "del" function (not a method) can be used to remove items from list.
- del(list_variable[index]) is used to remove item(s) of a list
- Difference between "pop" and "del":
  - "pop" returns the value of deleted item
  - "del" does not return any value

```
>>> L1=[2, 4.9, 5, 7]
>>> del(L1[2])
>>> L1
[2, 4.9, 7]
>>> del(L1[0])
>>> L1
[4.9, 7]
```

Deleting Items from Lists

- Slices can be used to delete a sublist
- If no index is given the variable is deleted

```
>>> L1=[2, 4.9, 5, 7, 11 , 123]
>>> del(L1[:4])
>>> L1
[11, 123]
>>> del(L1)
>>> L1
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.NameError: name 'L1' is not defined
```

Deleting Items from Lists

- If you know what to delete but do not know its index, “remove” method can be used
- Item to be removed is given as argument to “remove” method
- If there is no such item, it will cause an error

```
>>> L1=[2, 4.9, 5, 7, 11, 123]
>>> L1.remove(7)
>>> L1
[2, 4.9, 5, 11, 123]
>>> L1.remove(12)
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.ValueError: list.remove(x): x not in
list
```

“sorted” and “sort”

- “sorted” function or “sort” method can be used to sort the items of a list in an increasing order
- “sorted” sorts the list and returns a new sorted list. Original list items do not change
- “sort” method sorts the list item

```
>>> L1=[32, 14.9, 5, 11, 123]
>>> sorted(L1)
[5, 11, 14.9, 32, 123]
>>> L1
[32, 14.9, 5, 11, 123]
>>> L1.sort()
>>> L1
[5, 11, 14.9, 32, 123]
```

“sorted” and “sort”

- Lists of strings can also be sorted according to their Unicode numbers

```
>>> L2=['as','a','test','string','s1']
>>> sorted(L2)
['a', 'as', 's1', 'string', 'test']
```

“sorted” and “sort”

- Mixed item lists require more arguments (that we may cover later) to “sorted” or “sort”
- Otherwise, it gives an error as it does not know how to order strings and floats

```
>>> L1=[32, 14.9, 5, 11, 123, 'q','as']
>>> sorted(L1)
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.TypeError: unorderable types: str() <
float()
```

Summation of Items in a List

- "sum" function can be used to add the elements in a list if they are all numbers
- Otherwise, this will cause error

```
>>> L1=[32, 14.9, 5, 11, 123]
>>> L2=['as', 'a', 'test', 'string', 's1']

>>> sum(L1)
185.9
>>> sum(L2)
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

“split”

- “split” method is used to split a string into a list
- As an argument to “split” a delimiter is given
- If no argument is given, by default the delimiter is the white space
- Delimiter can be a character or it can be string
- If the delimiter cannot be found in the string, the whole string will be the only item of the list
- Delimiters are case sensitive

```
>>> S="This is a test string ; Another
  string ; New string"
>>> L1=S.split()
>>> L1
['This', 'is', 'a', 'test', 'string', ';',
  'Another', 'string', ';', 'New', 'string']
>>> L2=S.split(';')
>>> L2
['This is a test string ', ' Another string
  ', ' New string']
>>> L3=S.split('st')
>>> L3
['This is a te', ' ', 'ring ; Another ',
  'ring ; New ', 'ring']
```

“join”

- “join” is a method of string
- It joins the items of a list (with string items) by replacing the string in between the list items as delimiter

```
>>> L1=['This', 'is', 'a', 'test',
  'string',';', 'Another', 'string', ';',
  'New‘, 'string']
>>> S='*1*'
>>> S.join(L1)
'This*1*is*1*a*1*test*1*string*1*;*1*Another*
  1*string*1*;*1*New*1*string'
```

FOR LOOPS

For Loops

- “for” loops are used to iterate through the items of a list

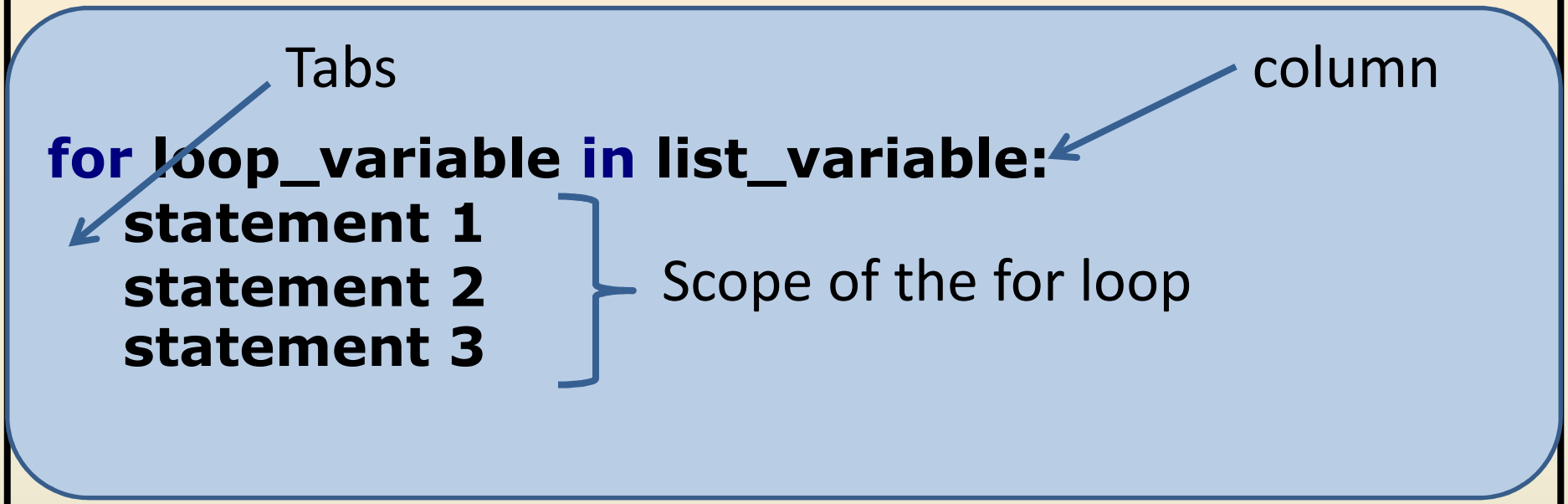
```
for s in ['Ali','Ayse','Fatma']:
    print('Hello ' + s)
```

- In each iteration of the loop, one item of the list is assigned to loop variable
- “for” loop will iterate as the number of items within list

```
Hello Ali
Hello Ayse
Hello Fatma
```

For Loops

- “for” loop structure



For Loop

```
for loop_variable in list_variable:
    statement 1
    statement 2
    statement 3
```

In the first iteration of the loop, first item of the list is assigned to loop variable

In the second iteration of the loop, second item of the list is assigned to loop variable

In the Nth iteration of the loop, Nth item of the list is assigned to loop variable

Example

- Add the square of numbers between 1 and 150

```
sum=0

for num in range(1,151):
    sum=sum+num**2

print('Sum of squares: ' + str(sum))
```

“else” in for loop

- “else” can be used in the for loop
- After all items within the list are used (in iteration len(list)+1), statements within the else scope are executed

```
for i in range(1, 5):
    print(i)
else:
    print('The for loop is over')
```

```
1
2
3
4
The for loop is over
```

“break” and “continue” in For Loops

- “break” and “continue” work within the for loops
- After “break” statement, the for loop is terminated. Scope of else is not executed

```
for i in range(1, 5):
   if (i==3):
      break
   print(i)
else:
   print('The for loop is over')
```

```
1
2
```

“break” and “continue” in For Loops

- After “continue” statement, the current iteration of the for loop is skipped.
- Next item of the list is assigned to loop variable
- Statements in the scope of “else” are executed when the items of the list are finished

“break” and “continue” in For Loops

```
for i in range(1, 5):
   if (i==3):
      continue
   print(i)
else:
   print('The for loop is over')
```

```
1
2
4
The for loop is over
```

Example

- Assume there is a list of strings,
- Each string is like

"student number;student name;midterm 1 grade;midterm 2 grade; final grade"

Example:

L=["040080111;Ayse Ucan;80;75;45",

"040080134;Ali Yaman;100;55;25",

"040080151;Hasan Kacan;8;45;15",

…

"040080312;Yaman Derin;90;33;77"]

Example

- Write a Python string that computes the weighted average grade such as

  0.2*midterm1+0.3*midterm2+0.5*final

- List all students and their average grades in the following format

  Number**Name**Average grade

Example

040080123**Ali Urgan**34.67

040080324**Ayse Ucan**68.5

```
L=['040080111;Ayse Ucan;80;75;45',
   '040080134;Ali Yaman;100;55;25',
   '040080151;Hasan Kacan;8;45;15',
   '040080312;Yaman Derin;90;33;77']

for line in L:
    student_info=line.split(';')
    student_number=student_info[0]
    student_name=student_info[1]
    midterm1_grade=int( student_info[2] )
    midterm2_grade=int( student_info[3] )
    final_grade=int( student_info[4] )

    # compute the average grade
    average_grade=0.2*midterm1_grade + \
                  0.3*midterm2_grade + \
                  0.5*final_grade

    # print the result
    tmp_list=[student_number,student_name,str(average_grade)]
    delimiter='**'
    print(delimiter.join(tmp_list))
```

Example

- Generate 1000 random integers between 0 and 5
- Count the number of occurrences (also called histogram) for each integer (ie. how many 3 in the 1000 random integers) and print

```
import random

hist=[0,0,0,0,0,0]

i=0;
# create 1000 random numbers
while i<1000:
    num=random.randint(0,5)
    i=i+1
    hist[num]=hist[num]+1

# print the histogram
for i in range(0,6):
    print(str(i) + ' --> ' + str(hist[i]))
```

Example

- Fibonacci series

  1,1,2,3,5,…
- First two numbers are 1
- After two numbers, each number is is the addition of the previous two numbers
- Write a Python code that prints the first 10 numbers in the Fibonacci series

```
i=0;
fibonacci=[]

for i in range(0,10):
    if i==0 or i==1:
        fibonacci.append(1)
    else:
        num=fibonacci[i-1]+fibonacci[i-2]
        fibonacci.append(num)

    print(str(i) + ' --> ' + str(fibonacci[i]))
```

0 --> 1
1 --> 1
2 --> 2
3 --> 3
4 --> 5
5 --> 8
6 --> 13
7 --> 21
8 --> 34
9 --> 55

Example

- Exponent of Euler number (e^x) can be computed as

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

- Write a Python code that gets x from user
- Computes e^x to the 1000th iteration
- Check

e^1=2.71828182845904523536028747135266249775572

```
i=0;
x=int(input('Enter the exponent of Euler: '))

sum=1
mult=1
fact=1

for i in range(1,10):
    mult=mult*x
    fact=fact*i
    sum = sum + mult/fact

print('e**x is ' + str(sum))
```