List (cont.) and Dictionary

01204111 Section 1

Task: *List Management*

- Write a program that interact with user with these commands.

```
add 3
add 1
add 2
print
items = 3 1 2
add 3
del 3
add 0
print
items = 1 2 0
exit
Bye.
```

Task: *List Management*

```
items = []
command = ''
while command != 'exit' :
        command = input()
        command_list = command.split()
        command = command_list[0]
        if len(command_list) == 1 :
                if command == 'print' :
                        print_list(items)
        elif len(command_list) == 2 :
                item = int(command_list[1])
                if command == 'add' :
                        items = add_to_list(items,item)
                if command == 'del' :
                        items = del_from_list(items,item)
print('Bye.')
```

Task: *List Management*

```
def print_list(items) :
        print('items = ',end='')
        for item in items :
                print(item,end=' ')
        print()
def add_to_list(items,item) :
        items.append(item)
        return items
def del_from_list(items,item) :
        new_items = []
        for tmp in items :
                if tmp != item :
                        new_items.append(tmp)
        return new_items
```

ASCII

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (**ASCII**) is a code for representing English characters as numbers. Each letter assigned a number from 0 to 127.

```
ord('A')
ord('a')
ord('0')
chr(98)
```

```
97
65
48
b
```

Code	Char	Code	Char	Code	Char	Code	Char
0	NUL	32	SPACE	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Task: *alphabet counter*

- Write a counting program with alphabet a-z.

```
1
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

```
5
a f k p u z
```

```
-1
z y x w v u t s r q p o n m l k j i h g f e d c b a
```

```
-7
z s l e
```

Task: *alphabet counter*

```python
step = int(input())
if step > 0 :
        for i in range(ord('a'), ord('z')+1, step):
                print(chr(i), end=' ')
else :
        for i in range(ord('z'), ord('a')-1, step):
                print(chr(i), end=' ')
```

ASCII

```
>>> one = 1
>>> one = str(one)
>>> one
'1'
>>> ord(one)
49
```

Change 1 to character '1'

```
>>> one = 1
>>> one = chr(one)
>>> one
'\x01'
>>> ord(one)
1
```

chr() returns the character in the first order

Task: *upper_case & lower_case*

- Write a function name **my_upper_case(string)** that convert all characters to uppercase.

```
Computer and Programming!!!
COMPUTER AND PROGRAMMING!!!
```

- Write a function name **my_lower_case(string)** that convert all characters to lowercase.

```
Computer and Programming!!!
computer and programming!!!
```

Task: *upper_case*

```
def my_upper_case(string):
        result = ''
        for i in string:
                if ord('a') <= ord(i) <= ord('z'):
                        result += chr(ord(i) - ord('a') + ord('A'))
                else:
                        result += i
        return result

print(my_upper_case('Computer and Programming!!!'))
```

```
COMPUTER AND PROGRAMMING!!!
```

Task: *lower_case*

```python
def my_lower_case(string):
        result = ''
        for i in string:
                if ord('A') <= ord(i) <= ord('Z'):
                        result += chr(ord(i) - ord('A') + ord('a'))
                else:
                        result += i
        return result

print(my_lower_case('Computer and Programming!!!'))
```

```
computer and programming!!!
```

Lowercase Uppercase

```
>>> string = 'Computer and Programming!!!'
>>> string
'Computer and Programming!!!'
>>> string.upper()
'COMPUTER AND PROGRAMMING!!!'
>>> string.lower()
'computer and programming!!!'
```

Count

- The **.count()** method adds up the number of times a character or sequence of characters appears in a string.

```
S = 'That that is is that that is
not is not is that it it is'
print(S.count('t'))
print(S.count('that'))
print(S.lower().count('t'))
```

```
13
4
14
```

Find

- We search for a specific character or characters in a string with the **.find()** method.

```
S = 'On the other hand, you have
different fingers.'
print(S.find('hand'))
print(S.find('o'))
print(S.find('o', 8))
print(S.find('e', 20, -5))
```

```
13
7
20
26
```

Replace

- Let's say we want to increase the value of a statement. We do so with the **.replace()** method.

```
S = 'I intend to live forever, or die trying.'
print(S.replace('to', 'three'))
print(S.replace('fore', 'five'))
```

```
I intend three live forever, or die trying.
I intend to live fivever, or die trying.
```

Dictionary Data Type

- Like a list, a **dictionary** is a collection of many values. But unlike indexes for lists, indexes for dictionaries can use many different datatypes. Indexes for dictionaries are called **keys** and a key with its associated value is called a **key-value pair**.

```
myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
print('My cat has ' + myCat['color'] + 'fur.')
```

```
My cat has gray fur.
```

Dictionary Order

- Unlike lists, items in **dictionaries** are unordered.

```
eggs1 = ['omelet', 'poached eggs', 'fried eggs']
eggs2 = ['fried eggs', 'omelet', 'poached eggs']
print(eggs1 == eggs2)
```

```
False
```

```
sudwork1 = {'fried chicken': 1, 'WingZ Zabb': 3, 'Nuggets': 2}
sudwork2 = {'WingZ Zabb': 3, 'Nuggets': 2, 'fried chicken': 1}
print(sudwork1 == sudwork2)
```

```
True
```

Dictionary Order

- Because **dictionaries** are unordered, the can't be sliced like lists.
- Trying to access a key that does not exist in dictionary will result in an error message.

```python
sudwork = {'fried chicken': 1, 'WingZ Zabb': 3, 'Nuggets': 2}
print(sudwork['zinger burger'])
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'zinger burger'
```

Dictionary Order

- To decrease the number of KeyError error in program, we also check **dictionary.keys()** before getting value.

```
sudwork = {'fried chicken': 1, 'WingZ Zabb': 3, 'Nuggets': 2}
if 'zinger burger' in sudwork.keys():
        print(sudwork['zinger burger'])
else:
        print(0)
```

```
0
```

Dictionary Order

- You can also use **get(x,y)** method to get the value in index x. if the index x is not exist it returns the value y.

```python
sudwork = {'fried chicken': 1, 'WingZ Zabb': 3, 'Nuggets': 2}
print(sudwork.get('fried chicken', 0))
print(sudwork.get('zinger burger', 0))
```

```
1
0
```

Task: *number to string*

- Write a program that change letter 0-9 to Zero-Nine

```
1999
One Nine Nine Nine
```

```
2017
Two Zero One Seven
```

```
1638
One Six Three Eight
```

```
007
Seven
```

Task: *number to string*

```
convert = {0:'Zero', 1:'One', 2:'Two', 3:'Three', 4:'Four',
5:'Five', 6:'Six', 7:'Seven', 8:'Eight', 9:'Nine'}
number = str(int(input()))
for i in number :
       i = int(i)
       if i in convert :
              print(convert[i], end=' ')
print()
```

.setdefault(x,y)

- You will often have to set a value in a dictionary for a certain key only if that key does not already have a value.

```
breakfast = {'fried eggs': 2, 'beacons': 5}
if 'hams' not in breakfast :
      breakfast['hams'] = 4
print(breakfast)
```

```
{'fried eggs': 2, 'beacons': 5, 'hams': 4}
```

```
breakfast = {'fried eggs': 2, 'beacons': 5}
breakfast.setdefault('hams', 4)
print(breakfast['hams'])
breakfast.setdefault('hams', 10)
print(breakfast['hams'])
```

```
4
4
```

Task: *dictionary of characters*

- Write a program that make a dictionary of number of existing characters.

```
You are what you eat.
{'y': 2, 'o': 2, 'u': 2, 'a': 3, 'r': 1, 'e': 2, 'w': 1, 'h': 1, 't': 2}
```

```
I'm Lovin' it
{'i': 3, 'm': 1, 'l': 1, 'o': 1, 'v': 1, 'n': 1, 't': 1}
```

```
Would you like some buns or shumai?
{'w': 1, 'o': 4, 'u': 4, 'l': 2, 'd': 1, 'y': 1, 'i': 2, 'k': 1, 'e': 2,
's': 3, 'm': 2, 'b': 1, 'n': 1, 'r': 1, 'h': 1, 'a': 1}
```

Task: *dictionary of exist characters*

```
count = {}
string = input().lower()
for c in string :
        if 'a' <= c <= 'z' :
                count.setdefault(c, 0)
                count[c] += 1
print(count)
```

Nested Dictionaries and Lists

- You may find you need dictionaries and lists that contain other dictionaries and lists. Lists are useful for associating keys with values.

```
myAquarium = [{'type': 'fish', 'species': 'neon' , 'nb': 12},
              {'type': 'fish', 'species': 'goldfish' , 'nb': 3},
              {'type': 'snail', 'species': 'horned nerite snail' , 'nb': 7},
              {'type': 'plant', 'species': 'hydrilla' , 'nb': 10},
              {'type': 'shrimp', 'species': 'dwarf shrimp' , 'nb': 50}]
```

Task: *myAquarium*

- From myAquarium list, write programs that answers these questions.
  - How many types of living things do you have? And what are they?
  - How many species of fish do you have? And what are they?
  - How many fish do you have?
  - Which species do you have most? And how many of them?
  - Show the species name that contains letter 'o'

Task: *myAquarium*

How many types of living things do you have? And what are they?

```
types = []
for tmp in myAquarium :
       tmpType = tmp['type']
       if tmpType not in types :
              types.append(tmpType)
print(len(types))
print(types)
```

```
4
['fish', 'snail', 'plant', 'shrimp']
```

Task: *myAquarium*

How many species of fish do you have? And what are they?

```
fishSpecies = []
for tmp in myAquarium :
        tmpType = tmp['type']
        if tmpType == 'fish' :
                fishSpecies.append(tmp['species'])
print(len(fishSpecies))
print(fishSpecies)
```

```
2
['neon', 'goldfish']
```

Task: *myAquarium*

How many fish do you have?

```
nbOfFish = 0
for tmp in myAquarium :
        tmpType = tmp['type']
        if tmpType == 'fish' :
                nbOfFish += tmp['nb']
print(nbOfFish)
```

```
15
```

Task: *myAquarium*

Which species do you have most? And how many of them?

```
mostNb = 0
mostSpecies = ''
for tmp in myAquarium :
        tmpNb = tmp['nb']
        if tmpNb > mostNb :
                mostNb = tmpNb
                mostSpecies = tmp['species']
print(mostSpecies)
print(mostNb)
```

```
Dwarf shrimp
50
```

Task: *myAquarium*

- Show the species name that contains letter 'o'

```
containO = []
for tmp in myAquarium :
        if 'o'  in tmp['species'] :
                containO.append(tmp['species'])
print(containO)
```

```
['neon', 'goldfish', 'horned nerite snail']
```

Task: *printing 2D list and transpose*

- Write function **print2dList(list)** and **printTranspose(list)** then test the function with grid.

- Transpose is an operation which flips a table over its diagonal, that is it switches the row and column indices of the table by producing another table.

```
grid = [['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]
```

Task: *printing 2D list and transpose*

```python
def print2dList(grid):
        for row in grid:
                for col in row:
                        print(col, end='')
                print()
```

```python
def print2dList(grid):
        for i in range(0, len(grid)):
                for j in range(0, len(grid[0])):
                        print(grid[i][j], end='')
                print()
```

```
......
.OO...
OOOO..
OOOOO.
.OOOOO
OOOOO.
OOOO..
.OO...
......
```

Task: *printing 2D list and transpose*

```
def printTranspose (grid):
        for j in range(0, len(grid[0])):
                for i in range(0, len(grid)):
                        print(grid[i][j], end='')
                print()
```

```
..OO.OO..
.OOOOOOO.
.OOOOOOO.
..OOOOO..
...OOO...
....O....
```

Task: *Add Matrices*

- Write a program to add m x n matrices provided by user.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} \dots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \dots & b_{1n} \\ b_{21} & b_{22} & b_{23} \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & b_{m3} \dots & b_{mn} \end{bmatrix} = \begin{bmatrix} (a_{11}+b_{11}) & (a_{12}+b_{12}) & (a_{13}+b_{13}) \dots & (a_{1n}+b_{1n}) \\ (a_{21}+b_{21}) & (a_{22}+b_{22}) & (a_{23}+b_{23}) \dots & (a_{2n}+b_{2n}) \\ \dots & \dots & \dots & \dots \\ (a_{m1}+b_{m1}) & (a_{m2}+b_{m2}) & (a_{m3}+b_{m3}) \dots & (a_{mn}+b_{mn} \end{bmatrix}$$

Task: *Add Matrices*

```
m = int(input('m: '))
n = int(input('n: '))
print('\nMatrix A')
matrix_a = get_matrix_from_user(m, n)
print('\nMatrix B')
matrix_b = get_matrix_from_user(m, n)
print()
result = add_matrix(matrix_a, matrix_b)
print_matrix(result)
```

```
m: 2
n: 3

Matrix A
Row 1: 1 2 3
Row 2: 4 5 6

Matrix B
Row 1: 7 8 9
Row 2: 10 11 12

|  8 10 12  |
| 14 16 18  |
```

Task: *Add Matrices*

```
m = int(input('m: '))
n = int(input('n: '))
print('\nMatrix A')
matrix_a = get_matrix_from_user(m, n)
print('\nMatrix B')
matrix_b = get_matrix_from_user(m, n)
print()
result = add_matrix(matrix_a, matrix_b
print_matrix(result)
```

```
def get_matrix_from_user(m, n):
    matrix = []
    for i in range(1, m + 1):
        user_input = input('Row {}: '.format(i))
        row = user_input.split(' ')
        for j in range(len(row)):
            row[j] = int(row[j])
        matrix.append(row)
    return matrix
```

Task: *Add Matrices*

```
m = int(input('m: '))
n = int(input('n: '))
print('\nMatrix A')
matrix_a = get_matrix_from_user(m, n)
print('\nMatrix B')
matrix_b = get_matrix_from_user(m, n)
print()
result = add_matrix(matrix_a, matrix_b)
print_matrix(result)
```

```
def add_matrix(a, b):
    result = []
    for i in range(len(a)):
        row = []
        for j in range(len(a[i])):
            row.append(a[i][j] + b[i][j])
        result.append(row)
    return result
```

Task: *Add Matrices*

```
m = int(input('m: '))
n = int(input('n: '))
print('\nMatrix A')
matrix_a = get_matrix_from_user(m, n)
print('\nMatrix B')
matrix_b = get_matrix_from_user(m, n)
print()
result = add_matrix(matrix_a, matrix_b)
print_matrix(result)
```

```
def print_matrix(matrix):
    for i in range(len(matrix)):
        print('|', end='')
        for j in range(len(matrix[i])):
            print('{:>3}'.format(matrix[i][j]), end='')
        print('{:>3}'.format('|'))
```

Task: *Multiply Matrices*

- Write a function to multiply 2 matrices provided by user.
- Assume that user provides m x n and p x q matrices respectively when n = p.
- Result will be m x q matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}$$

Task: *Multiply Matrices*

- From matrix A x B = C
- You can see that C = $[c_{ij}]$ when $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}$
- $0 < i < m$ and $0 < j < q$
- But do not forget that the first index of list is 0.

$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}$$

Task: *Multiply Matrices*

```python
def multiply_matrix(a, b):
    result = []
    for i in range(len(a)):
        row = []
        for j in range(len(b[i])):
            c = 0
            for n in range(len(a[i])):
                c += a[i][n] * b[n][j]
            row.append(c)
        result.append(row)
    return result
```

```
Matrix A
|  1  2  3  |
|  4  5  6  |

Matrix B
|  1  1  |
|  2  3  |
|  5  0  |

Result
| 20  7  |
| 44 19  |
```

References

- Python Slides 2017 **– Department of Computer Engineering Kasetsart University**
- Think Python **– Allen B. Downey**
- https://automatetheboringstuff.com/chapter5/
- https://thehelloworldprogram.com/python/python-string-methods/