

CSIT110

Fundamental Programming with Python

List and Tuple

Goh X. Y.



In this lecture

- Recap
 - List
 - Splicing
 - List methods
 - Tuple
 - Multi-dimensional list
- List Comprehension

List

This is how we define a list:

```
list_variable = [item1, item2, ..., itemN]
```



List

A list/array is used to hold a list of items:

```
animal_list = ["dog", "cat", "frog"]
```

```
fibonacci_numbers = [0, 1, 1, 2, 3, 5, 8, 13]
```

```
prime_numbers = [2, 3, 5, 7, 11, 13, 17]
```

```
subject_list = ["MATH101", "CS222", "PHY102", "ACCY203"]
```

```
correct_answer_list = [True, False, True, True, False]
```

```
random_list = ["Production Info", 342, False]
```

```
selected_products = [] # this is an empty list
```

List

List is zero-indexed

List items can be accessed via **index/indices**:

```
animal_list = ["dog", "cat", "frog"]
```

```
print(animal_list[0])    → "dog"  
print(animal_list[1])    → "cat"  
print(animal_list[2])    → "frog"
```

```
fibonacci_numbers = [0, 1, 1, 2, 3, 5, 8, 13]
```

```
print(fibonacci_numbers[0]) → 0  
print(fibonacci_numbers[1]) → 1  
print(fibonacci_numbers[2]) → 1  
print(fibonacci_numbers[3]) → 2  
print(fibonacci_numbers[4]) → 3  
print(fibonacci_numbers[5]) → 5  
print(fibonacci_numbers[6]) → 8  
print(fibonacci_numbers[7]) → 13
```

List - Splicing

Sub-lists can be retrieved via **indices**:

```
a[start:stop] # elements from start to stop-1
a[start:]    # elements from start to the rest of the array
a[:stop]     # elements from the beginning to stop-1
a[:]         # a copy of the whole array
```

Just like string! 

List – Length of list

using `len` to find out how many items in the list:

```
animal_list = ["dog", "cat", "frog"]
```

```
animal_count = len(animal_list) → 3
```

Note that `len(animal_list)` is 3, but the **last index** is 2 because the index start at 0.

```
print(animal_list[0]) → "dog"  
print(animal_list[1]) → "cat"  
print(animal_list[2]) → "frog"
```

List

```
animal_list = ["dog", "cat", "frog"]
```

```
print(animal_list[0])    → "dog"
```

```
print(animal_list[1])    → "cat"
```

```
print(animal_list[2])    → "frog"
```

We can go through the list using for loop via **index**:

```
for i in range(0, len(animal_list)):  
    print(animal_list[i])
```

Or:

```
for i in range(0, len(animal_list)):  
    animal = animal_list[i]  
    print(animal)
```


List

Example – increase each item by 10

```
random_numbers = [1, 4, 4, 10, -1]
```

Using for-loop, increase each item by 10:

```
for i in range(0, len(random_numbers)):  
    random_numbers[i] = random_numbers[i] + 10
```

```
print(random_numbers)    → [11, 14, 14, 20, 9]
```

List – An iterable Object

```
animal_list = ["dog", "cat", "frog"]
```

```
print(animal_list[0])    → "dog"  
print(animal_list[1])    → "cat"  
print(animal_list[2])    → "frog"
```

Alternative way: go through the list using for loop - without a counter

```
for animal in animal_list:  
    print(animal)
```

Some List Methods

`[] .append()`

`[] .insert()`

`[] .remove()`

`[] .count()`

`[] .index()`

`[] .sort()`

`[] .reverse()`

`[] .clear()`

`[] .extend()`

List – Update list element

```
animal_list = ["dog", "cat", "frog"]
```

```
animal_list[0] = "wombat"
```

```
animal_list[1] = "echidna"
```

```
animal_list[2] = "koala"
```

```
animal_list[3] = "kangaroo" ERROR: index out of range
```

```
# we have to do this instead
```

```
animal_list.append("kangaroo")
```

```
animal_list.append("emu")
```

```
print(animal_list)
```

```
→ ['wombat', 'echidna', 'koala', 'kangaroo', 'emu']
```

List

items **appended** are added to the end of the list:

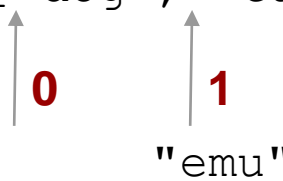
```
fibo_numbers = [0, 1, 1, 2, 3, 5, 8, 13]
fibo_numbers.append(21)
fibo_numbers.append(34)
fibo_numbers.append(55)
fibo_numbers.append(89)
print(fibo_numbers)
```

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

List

items can be **inserted** into the list:


```
animal_list = ["dog", "cat", "frog"]
```



The diagram shows the list `["dog", "cat", "frog"]` with indices `0` and `1` marked below the first two elements. An upward arrow from index `1` points to the word `"emu"`, indicating its insertion at that position.

```
animal_list.insert(1, "emu")
```

```
["dog", "emu", "cat", "frog"]
```



The diagram shows the updated list `["dog", "emu", "cat", "frog"]` with indices `0`, `1`, `2`, and `3` marked below the first four elements. An upward arrow from index `3` points to the word `"koala"`, indicating its insertion at that position.

```
animal_list.insert(3, "koala")
```

```
["dog", "emu", "cat", "koala", "frog"]
```

List – Remove by index

items can be **deleted** from the list via **index**:

```
subject_list = ["MATH101", "CS222", "PHY102", "ACCY203"]
```

↑ **0** ↑ **1**
del

```
# deleting the item at index 1
```

```
del subject_list[1]
```

```
["MATH101", "PHY102", "ACCY203"]
```

↑ **0** ↑ **1** ↑ **2**
del

```
# deleting the item at index 2
```

```
del subject_list[2]
```

```
["MATH101", "PHY102"]
```

List – Remove by value

items can be **removed** from the list via **value**, only the **first appearance** get removed.

```
random_numbers = [3, 12, 4, 5, 4, 3, 2, 6, 12]
```

```
# remove the first appearance of 4
```

```
random_numbers.remove(4)
```

```
# → [3, 12, 5, 4, 3, 2, 6, 12]
```

```
# remove the first appearance of 12
```

```
random_numbers.remove(12)
```

```
# → [3, 5, 4, 3, 2, 6, 12]
```

```
# remove the first appearance of 7
```

```
random_numbers.remove(7)
```

```
ValueError: list.remove(x): x not in list
```


List – Count elements

```
random_numbers = [1, 4, 4, 10, -1]
```

count how many an item appears in the list

```
four_count = random_numbers.count(4) → 2
```

```
ten_count = random_numbers.count(10) → 1
```

```
five_count = random_numbers.count(5) → 0
```

List – Search element, getting index of the first

```
random_numbers = [1, 4, 4, 10, -1]
```

find the smallest **index** of an item in the list

```
four_index = random_numbers.index(4) → 1
```

```
ten_index = random_numbers.index(10) → 3
```

```
five_index = random_numbers.index(5)  
ValueError: 5 is not in the list
```

Get **FIRST** occurrence of – List vs String

```
myList = ["apple", "banana", "cherry", "apple"]  
myList.index("apple")
```

```
myText = "I have a fruit basket"  
myText.find('a')
```



Both returns the index of the **FIRST** occurrence if the element exists.

If the element does not exist, the list method raise an error, while the string method returns -1

Get LAST occurrence – List vs String

```
myList = ["apple", "banana", "cherry", "apple"]  
index = len(myList) - 1 - myList[::-1].index('apple')
```

```
myText = "I have a fruit basket"  
index = myText.rfind('a')
```



.rfind() to get the **LAST** occurrence of the character

List – Find smallest and largest numeric element

```
random_numbers = [1, 4, 4, 10, -1]
```

```
# finding min item
```

```
number_min = min(random_numbers) → -1
```

```
# finding max item
```

```
number_max = max(random_numbers) → 10
```

List – Sorting

```
random_numbers = [1, 4, 4, 10, -1]
```

Sorting a list and return a new list, **original list is unchanged**

```
sorted_numbers = sorted(random_numbers)
```

sorted() is a built-in function  that returns a new list

Now sorted_numbers is [-1, 1, 4, 4, 10]

but random_numbers is unchanged:


random_numbers is still [1, 4, 4, 10, -1]

List – Sorting

```
random_numbers = [1, 4, 4, 10, -1]
```

Sorting a list and **modify the original list**

```
random_numbers = [1, 4, 4, 10, -1]  
random_numbers.sort()
```

list.sort() is a method of the  list object which modifies said list

```
now random_numbers is changed,  
random_numbers is now [-1, 1, 4, 4, 10]
```

List – reverse order and clear

```
random_numbers = [1, 4, 4, 10, -1]
```

items can be reversed

```
random_numbers.reverse()    # now [-1, 10, 4, 4, 1]
```

remove all items

```
random_numbers.clear()      # now []
```


List – adding and multiplying

```
list1 = [1, 4, 4, 10, -1]  
list2 = [10, 7, 5]
```

adding two lists

```
list12 = list1 + list2    # now list12 = [1, 4, 4, 10, -1, 10, 7, 5]
```

```
list21 = list2 + list1    # now list21 = [10, 7, 5, 1, 4, 4, 10, -1]
```

multiply a list

```
list3 = [9, 8]
```

```
list4 = list3 * 3          # now list4 = [9, 8, 9, 8, 9, 8]
```

List – extending the list

```
list1 = [1, 4, 4, 10, -1]  
list2 = [10, 7, 5]
```

Using `.append()`

```
list1.append(list2)  
# list1 is now [1, 4, 4, 10, -1, [10, 7, 5]]
```

Using `.extend()` instead

```
list1.extend(list2)  
# list1 is then [1, 4, 4, 10, -1, 10, 7, 5]
```

Example – Square sequence 0, 1, 4, 9, ...

Create a list and put the first 10 squares into the list

```
# initially, create an empty list
square_list = []
for i in range(0, 10):
    # adding square numbers to the list
    square_number = i * i
    square_list.append(square_number)
print("First 10 square numbers:")
print(square_list)
```

```
First 10 square numbers:
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Example – Fibonacci 0, 1, 1, 2, 3, 5, 8, ...

Create a list and put the first 10 fibonacci numbers into the list

```
fibonacci_list = []  
  
fibonacci_list.append(0)  
  
fibonacci_list.append(1)  
  
for i in range(2, 10):  
    fibo = fibonacci_list[i-1] + fibonacci_list[i-2]  
    fibonacci_list.append(fibo)
```

```
i=2  
fibo = fibonacci_list[1] + fibonacci_list[0] = 1 + 0 = 1  
fibonacci_list.append(fibo)  
i=3  
fibo = fibonacci_list[2] + fibonacci_list[1] = 1 + 1 = 2  
fibonacci_list.append(fibo)  
i=4  
fibo = fibonacci_list[3] + fibonacci_list[2] = 2 + 1 = 3  
fibonacci_list.append(fibo)  
i=5  
fibo = fibonacci_list[4] + fibonacci_list[3] = 3 + 2 = 5  
fibonacci_list.append(fibo)  
...
```

```
[]  
  
[0]  
  
[0, 1]  
  
i = 2  
[0, 1, 1]  
  
i = 3  
[0, 1, 1, 2]  
  
i = 4  
[0, 1, 1, 2, 3]  
  
...
```

Example

During winter break, each student can choose exactly one intensive subject to study. Write a program to

- **Step 1:** *let a student select a number of preferred subjects;*
- **Step 2:** *then among the preferred subjects the student selected, choose a random subject for student enrolment.*

Example

- **Step 1:** *let a student select a number of preferred subjects;*

```
# create a list of preferred subject, start with an empty list
subject_list = []

# repeatedly ask the user to enter subject code
while True:
    subject = input("Enter preferred subject code (enter QUIT to
quit): ")

    if(subject == "QUIT"):
        break

    # add subject to subject list
    subject_list.append(subject)

# display subjects the user has entered
print("You have chosen: " + str(subject_list))
```

Example

- *Step 1: let a student select a number of preferred subjects;*

```
Enter preferred subject code (enter QUIT to quit): MATH300  
Enter preferred subject code (enter QUIT to quit): COMP222  
Enter preferred subject code (enter QUIT to quit): ACCY100  
Enter preferred subject code (enter QUIT to quit): BUSS200  
Enter preferred subject code (enter QUIT to quit): QUIT  
You have chosen: ['MATH300', 'COMP222', 'ACCY100', 'BUSS200']
```

Example

- **2:** then among the preferred subjects the student selected, **choose a random subject** for student enrolment.

How can we choose a random subject?

Subject list: ['MATH300', 'COMP222', 'ACCY100', 'BUSS200']

Index:



We need to choose a random list index:

The index is a random number from **0** to **`len(subject_list) - 1`**

Example

- ***2: then among the preferred subjects the student selected, choose a random subject for student enrolment.***

```
# choose a random index from 0 to len(subject_list)-1
random_index = random.randint(0, len(subject_list)-1)
random_subject = subject_list[random_index]

# display the random subject enrolled for the user
print("You have been approved to enrol into " + random_subject)
```

```
You have chosen: ['MATH300', 'COMP222', 'ACCY100', 'BUSS200']
You have been approved to enrol into ACCY100
```

```
# import random module
import random ← remember to import random module at the top of the code

# create a list of preferred subject, start with an empty list
subject_list = []

# repeatedly ask the user to enter subject code
while True:
    subject = input("Enter preferred subject code (enter QUIT to quit): ")
    if(subject == "QUIT"):
        break

    # add subject to subject list
    subject_list.append(subject)

# display subjects the user has entered
print("You have chosen: " + str(subject_list))

# choose a random index from 0 to len(subject_list)-1
random_index = random.randint(0, len(subject_list)-1)

random_subject = subject_list[random_index]

# display the random subject enrolled for the user
print("You have been approved to enrol into " + random_subject)
```

Tuple

A tuple is similar to list but:

- A list can be changed
- A tuple is fixed

```
animal_list = ["dog", "cat", "frog"]
```

```
animal_tuple = ("dog", "cat", "frog")
```

```
# we can change list
```

```
animal_list[0] = "elephant"
```

```
# but we canNOT change tuple
```

```
animal_tuple[0] = "elephant" ERROR
```

Two-dimensional list

```
list2d = [  
    [1, 2, 3, 4],  
    [9, 8, 7, 6]  
]
```

```
print(list2d[0][1])
```

```
print(list2d[0][2])
```

```
print(list2d[1][0])
```

```
print(list2d[1][3])
```

`list2d[0]` → [1, 2, 3, 4]

`list2d[0][1]` → 2

2

3

9

6

Two-dimensional list

```
list2d = [  
    [1, 2, 3, 4],  
    [9, 8, 7, 6]  
]
```

```
print(list2d[0][1])
```

2

```
print(list2d[0][2])
```

3

```
print(list2d[1][0])
```

9

```
print(list2d[1][3])
```

6

`list2d[1]` → [9, 8, 7, 6]

`list2d[1][3]` → 6

Euler's magic square

68^2	29^2	41^2	37^2
17^2	31^2	79^2	32^2
59^2	28^2	23^2	61^2
11^2	77^2	8^2	49^2

Sum of numbers on each row, each column, and each diagonal is the same!

Euler's magic square

```
euler = [  
    [68**2, 29**2, 41**2, 37**2],  
    [17**2, 31**2, 79**2, 32**2],  
    [59**2, 28**2, 23**2, 61**2],  
    [11**2, 77**2, 8**2, 49**2]  
]
```

row sums

```
row1 = euler[0][0] + euler[0][1] + euler[0][2] + euler[0][3]  
row2 = euler[1][0] + euler[1][1] + euler[1][2] + euler[1][3]  
row3 = euler[2][0] + euler[2][1] + euler[2][2] + euler[2][3]  
row4 = euler[3][0] + euler[3][1] + euler[3][2] + euler[3][3]
```

column sums

```
column1 = euler[0][0] + euler[1][0] + euler[2][0] + euler[3][0]  
column2 = euler[0][1] + euler[1][1] + euler[2][1] + euler[3][1]  
column3 = euler[0][2] + euler[1][2] + euler[2][2] + euler[3][2]  
column4 = euler[0][3] + euler[1][3] + euler[2][3] + euler[3][3]
```

diagonal sums

```
diagonal1 = euler[0][0] + euler[1][1] + euler[2][2] + euler[3][3]  
diagonal2 = euler[0][3] + euler[1][2] + euler[2][1] + euler[3][0]
```

68^2	29^2	41^2	37^2
17^2	31^2	79^2	32^2
59^2	28^2	23^2	61^2
11^2	77^2	8^2	49^2

Euler's magic square

```
euler = [  
    [68**2, 29**2, 41**2, 37**2],  
    [17**2, 31**2, 79**2, 32**2],  
    [59**2, 28**2, 23**2, 61**2],  
    [11**2, 77**2, 8**2, 49**2]  
]
```

68^2	29^2	41^2	37^2
17^2	31^2	79^2	32^2
59^2	28^2	23^2	61^2
11^2	77^2	8^2	49^2

```
print("row1={0}, row2={1}, row3={2}, row4={3}"  
      .format(row1, row2, row3, row4))
```

```
print("column1={0}, column2={1}, column3={2}, column4={3}"  
      .format(column1, column2, column3, column4))
```

```
print("diagonal1={0}, diagonal2={1}"  
      .format(diagonal1, diagonal2))
```

```
row1=8515, row2=8515, row3=8515, row4=8515  
column1=8515, column2=8515, column3=8515, column4=8515  
diagonal1=8515, diagonal2=8515
```


Extra: Tuple (not tested)

- Unchanged and immutable
- Written with round brackets
- Otherwise very similar to <class 'list'>

```
x = ("apple", "banana", "cherry")
```



Extra: Tuple (not tested)

- Unchanged and immutable
- Written with round brackets
- Very similar to <class 'list'>

```
x = ("apple", "banana", "cherry")
```

- Workaround to change a tuple:

```
# tuple -> convert to list -> change list-> convert to tuple
```

```
x = ("apple", "banana", "cherry")
```

```
y = list(x)
```

```
y[1] = "kiwi"
```

```
x = tuple(y)
```

Warning: List is mutable!

Do not assign a list like this:

```
list1 = ['1', '2', '3']
```

```
list2 = list1
```

Do this instead

```
list2 = list1.copy()
```

Mutable data types in Python

- list, dictionary, user-defined classes

Immutable data types in Python

- int, float, decimal, bool, string, tuple, and range

Warning: List is mutable!

Try this!

```
list1 = ['1', '2', '3']  
list2 = list1  
list3 = list1.copy()  
list4 = list1  
list3[0] = '5'  
print(list1)  
print(list2)  
list2[0] = '0'  
print(list1)  
print(list4)
```

id()

Returns the “identity” of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime

Try this!

```
x = 300
print(id(x))
x += x
print(x)
print(id(x))

y = "hi"
print(id(y))
y += y
print(y)
print(id(y))

z = ["hi"]
print(id(z))
z += z
print(z)
print(id(z))
```

```
1969821108720
600
1969815567280
1969821022768
hihi
1969821333168
1969821022976
['hi', 'hi']
1969821022976
```

Note!

Basic Data Types are immutable

Collections are mutable

Pythonic

Pythonic means code that doesn't just get the syntax right but that follows the conventions of the Python community and uses the language in the way it is intended to be used. – a StackOverflow user

F-string (literal string interpolation)

List comprehension

for ... in

List comprehension (not tested by useful)

Instead of

```
squares = []  
for x in range(10):  
    squares.append(x**2)
```

Do

```
squares = [x**2 for x in range(10)]
```

List comprehension

More example:

<https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions>

Any questions?