LECTURE 8

DOHYUNG KIM

WHAT IS DISCUSSED IN THE LAST CLASS

- File I/O

TODAY, WE WILL LEARN ABOUT

- List

LIST

- Used to group values together
- Can contain even items of different types
- Created by square brackets

```
a = [1, 2, 3]
b = list(range(4))
c = ["string value", 20, True]
```

LIST

- **Empty list** can be created in two different ways

```
a = [ ]
b = list()

print(type(a), len(a), a)
print(type(b), len(b), b)
print(a == b)
```

- List with elements

```
a = [1, 2, 3]
b = ["hi"]
```

LIST

- Variable-length List

```python
n = 10
a = [0] * n
b = list(range(n))

print(type(a), len(a), a)
print(type(b), len(b), b)
```

- c.f.) List to/from string

```python
s1 = "Python programming"
l = list(s1)
s2 = "".join(l)

print(type(s1), len(s1), s1)
print(type(l), len(l), l)
print(type(s2), len(s2), s2)
print(s1==s2)
```

- String can be converted to list using **list()** function
- List can be converted to string using **join()** method

FUNCTIONS FOR LIST

- **len(), max(), min(), sum()**

```
l1 = [1, 2, 3, 4]
l2 = ["abcd", "cd", "ef", "ghi"]

print(len(l1))
print(max(l1))
print(min(l1))
print(sum(l1))

print(len(l2))
print(max(l2))
print(min(l2))
print(sum(l2))
```

INDEXING / SLICING IN LISTS

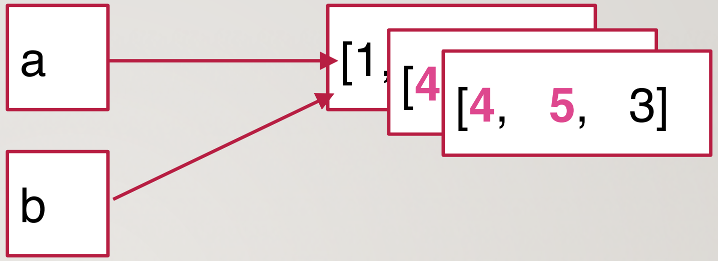
- The same as the manipulation of strings

```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

print("a          =", a)

print("a[0]       =", a[0])
print("a[2]       =", a[2])

print("a[-1]      =", a[-1])
print("a[-3]      =", a[-3])

print("a[0:2]     =", a[0:2])
print("a[1:4]     =", a[1:4])
print("a[1:6:2] =", a[1:6:2])
```

LIST ALIASING

- List Aliasing : points to the same list object

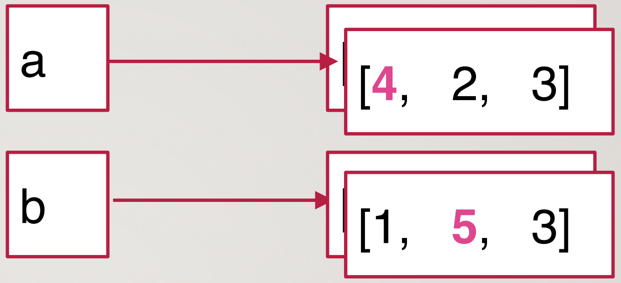
```
a = [1, 2, 3]
b = a

print(a is b)
print(a == b)

a[0] = 4
b[1] = 5

print(a)
print(b)
```



LIST ALIASING

- Same values but different list objects

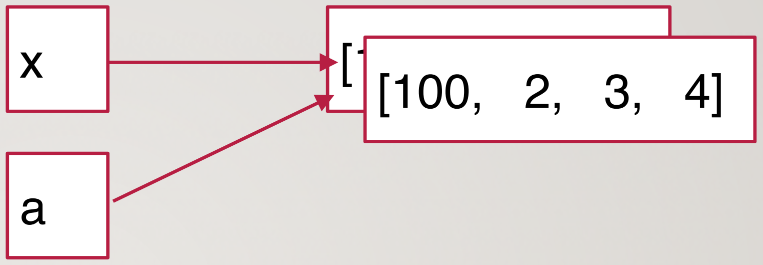
```
a = [1, 2, 3]
b = [1, 2, 3]

print(a is b)
print(a == b)

a[0] = 4
b[1] = 5

print(a)
print(b)
```



LIST ALIASING

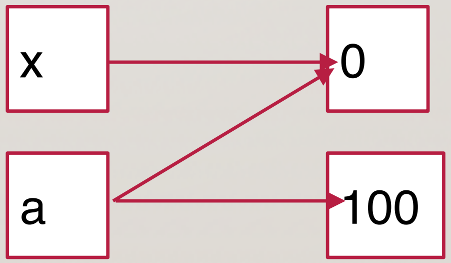
- List as a function parameter

```python
def funcWithList(a):
  a[0] = 100

x = [1, 2, 3, 4]
print(x)
funcWithList(x)
print(x)
```



- c.f.) Integer as a function parameter

```python
def funcWithInt(a):
  a = 100

x = 0
print(x)
funcWithInt(x)
print(x)
```



FINDING ELEMENTS IN LIST

- **in, not in, list.count()**

```
a = [1, 2, 3, 4, 1, 3, 3, 5, 7 ]
print("a      =", a)

print("2 in a =", (2 in a))
print("6 in a =", (6 in a))

print("2 not in a =", (2 not in a))
print("6 not in a =", (6 not in a))

print("frequency of 1: ", a.count(1))
print("frequency of 3: ", a.count(3))
print("frequency of 6: ", a.count(6))
```

FINDING ELEMENTS IN LIST

- **list.index()**

```
a = [1, 2, 3, 4, 1, 3, 3, 5, 7 ]
print("a      =", a)

print("index of 7 : ", a.index(7))
print("index of 4 : ", a.index(4))
print("index of 1 that comes from 2nd element: ", a.index(1, 2))
print("index of 3 that comes from 5th element: ", a.index(3, 5))
```

- There would be an **error** if you **call index() for the item not in the list**
  - Check before calling list.index()

```
if(8 in a):
  print("index of 8 : ", a.index(8))
```

ADDING ELEMENT TO LIST

- Remind that **string object is immutable**
- **List object is mutable**, that is, list object can be modified
  - e.g., add, remove, change elements in the list
- Items can be added using
  - **list.append()**
  - **list += list2**
  - **list.extend(list2)**
  - **list.insert(index, item)**

```
a = [1, 2]
print(a)

a.append("a")
print(a)

a += [3, 4]
print(a)

a.extend(["c", "d"])
print(a)

a.insert(3, "b")
print(a)
```

ADDING ELEMENT TO LIST

- Add elements non-destructively
  - Not changing the list, but creating a new one

```
a = [1, 2]

b = a + ["b"]
print(a)
print(b)

c = a[:2] + ["c"] + a[2:]
print(c)
```

REMOVING ELEMENT FROM LIST

- **list.remove()**

```
a = [2, 3, 5, 7, 9, 11, 13, 17]
print("a =", a)

a.remove(5)
print("After a.remove(5), a=", a)

a.remove(5)
print("After another a.remove(5), a=", a)
```

- With slice assignment

```
a = [2, 3, 5, 7, 9, 11, 13, 17]
a[2:4] = [ ]
print("a =", a)
```

- With **del** operator

```
a = [2, 3, 5, 7, 9, 11, 13, 17]
del a[2:4]
print("a =", a)
```

REMOVING ELEMENT FROM LIST

- **list.pop()**

```
a = [2, 3, 5, 7, 9, 11, 13, 17]
print("a =", a)
print()

item = a.pop(4)
print("After item = a.pop(4)")
print("   4th item : ", item)
print("   a =", a)
print()

item = a.pop(4)
print("After another item = a.pop(4)")
print("   4th item : ", item)
print("   a =", a)
print()
```

```
item = a.pop()
print("After item = a.pop()")
print("   the last item =", item)
print("   a =", a)
```

REMOVING ELEMENT FROM LIST

- Removing in a non-destructive way

```
a = [2, 3, 5, 7, 9, 11, 13, 17]
print("a =", a)
print()
b = a[:2] + a[3:]
print("After b = a[:2] + a[3:]")
print()
print("a =", a)
print("b =", b)
```

SWAPPING ELEMENT

- First try (failed)

```
a = [2, 3, 5, 7]
print("a =", a)
a[0] = a[1]
a[1] = a[0]
print("after swapping")
print("a=",a)
```

- Second try (using a temp variable)

```
a = [2, 3, 5, 7]
print("a =", a)
temp = a[1]
a[1] = a[0]
a[0] = temp
print("after swapping")
print("a =",a)
```

- Wow (using tuple assignment)

```
a = [2, 3, 5, 7]
print("a =", a)
a[0], a[1] = a[1], a[0]
print("after swapping")
print("a =",a)
```

LOOPING OVER LIST

- **for item in list**

```
a = [2, 3, 5, 7]
print(a, "\n")
print("Here are the items in a:")
for item in a:
  print(item, end=" ")
```

- **for index in range(len(list))**

```
a = [2, 3, 5, 7]
print(a, "\n")
print("Here are the items in a:")
for idx in range(len(a)):
  print(a[idx], end=" ")
```

PRACTICE

- Print out all elements of the list in reverse order

```
a = [2, 3, 5, 7]
print(a, "\n")
print("Here are the items of a in reverse order:”)
...
```

LOOPING OVER LIST

- **Looping backward**

```python
a = [2, 3, 5, 7]
print(a, "\n")
print("Here are the items of a in reverse order:")
for idx in range(len(a)-1, -1, -1):
  print(a[idx], end=" ")
```

- **Using reversed(list)**

```python
a = [2, 3, 5, 7]
print(a, "\n")
print("Here are the items of a in reverse order:")
for item in reversed(a):
  print(item, end=" ")
```

LOOPING OVER LIST

- **list.reverse()**

```
a = [2, 3, 5, 7]
print(a, "\n")
print("Here are the items of a in a reverse order:")
a.reverse()
for item in a:
  print(item, end=" ")
```

- Is it okay to merge a.reverse() and for-loop?

```
a = [2, 3, 5, 7]
print(a, "\n")
print("Here are the items of a in a reverse order:")
for item in a.reverse():
  print(item, end=" ")
```

LOOPING OVER LIST

- What would be the problem in this code?

```
a = [2, 3, 5, 7, 3]
print("a =", a)
print()

for idx in range(len(a)):
  if (a[idx] == 3):
    a.pop(idx)
    print(a)

print("error!")
```

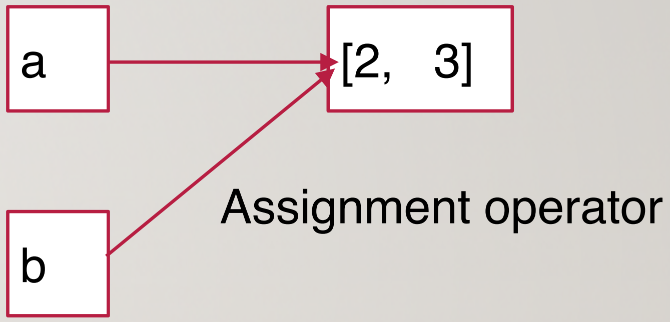
COMPARING LISTS

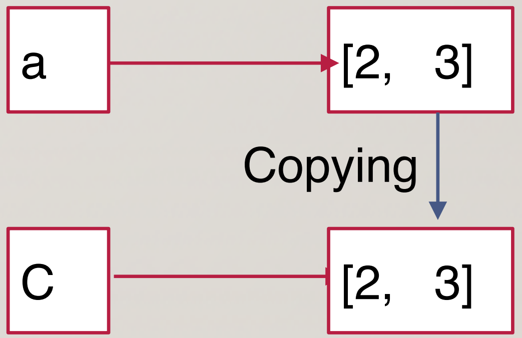
```python
a = [2, 3, 5, 7]
b = [2, 3, 5, 7]
c = [2, 3, 5, 8]
d = [2, 3, 5]

print("------------------")
print("a == b", (a == b))
print("a == c", (a == c))
print("a != b", (a != b))
print("a != c", (a != c))

print("------------------")
print("a < c", (a < c))
print("a < d", (a < d))
```

COPYING LIST

- Copying vs Aliasing

```
import copy

a = [2, 3]
b = a                # alising
c = copy.copy(a)     # copying

print("    a =", a)
print("    b =", b)
print("    c =", c)

a[0] = 5
print("After modifying")
print("    a =", a)
print("    b =", b)
print("    c =", c)
```





COPYING LIST

- Ways to do copying lists

```
import copy

a = [2, 3]
b = copy.copy(a)
c = a[:]
d = a + [ ]
e = list(a)
f = copy.deepcopy(a)
g = sorted(a)

a[0] = 4
print("a: ", a, "\nb: ", b,"\nc: ", c,"\nd: ",\
 d,"\ne: ", e, "\nf: ",  f,"\nb: ", g)
```

SORTING LIST

- **list.sort()** (sort list in a destructive way)
  - list.sort() does **NOT RETRUN** a list (similar to list.reserve())

```
a = [3, 2, 5, 11, 7]
print("a =", a)
a.sort()
print("After a.sort(), a =",a)
```

- **sorted(list)**, if you want to have a new list object having sorted elements of a list

```
a = [3, 2, 5, 11, 7]
print("a =", a)
print("----------------------")
b = sorted(a)
print("After b = sorted(a)")
print("a =", a)
print("b =", b)
```

SORTING LIST

- You can designate a **key** function

- list.sorted(key)

```
import copy
a = [-3, 2, -5, -11, 7]
b = copy.copy(a)
print("a =", a)
print("----------------------")
a.sort()
print("After a.sort()")
print("a =", a)
b.sort(key=abs)
print("After b.sort(key=abs)")
print("b =", b)
```

- sorted(list, key)

```
a = [-3, 2, -5, -11, 7]
print("a =", a)
print("----------------------")
print("After sorted(a)")
print("a =", sorted(a))
print("After sorted(a, key=abs)")
print("a =", sorted(a, key=abs))
```

* By the way, does this code work?

```
a = [-3, 2, -5, -11, 7]
print("a =", a)
print("----------------------")
print("After sorted(a)")
print("a =", a.sort(key=abs))
```

SORTING LIST

- sort a list in reverse order using **reserve** parameter

*** list.sort(reversed=True)**

```
import copy
a = [-3, 2, -5, -11, 7]
b = copy.copy(a)
print("a =", a)
print("---------------------")
a.sort(key=abs)
print("After a.sort(key=abs)")
print("a =", a)
b.sort(key=abs, reverse=True)
print("After b.sort(key=abs, reverse=True)")
print("b =", b)
```

*** sorted(list, reversed=True)**

```
a = [-3, 2, -5, -11, 7]
print("a =", a)
print("---------------------")
print("After sorted(a, key=abs, reversed=True)")
print("a =", sorted(a, key=abs, reverse=True))
```

CALL BY VALUE VS REFERENCE

- What would be the result?

```python
def func(a, val):
  for i in range(len(a)):
    a[i] = val
  val = 20

a = [1, 2, 3, 4, 5]
val = 10
print("a =", a, "val =", val)
func(a, val)
print("After func(a, val)")
print("a =", a, "val =", val)
```

PRACTICE

- What are the difference between removeAll1 and removeAll2

```
import copy

def removeAll1(a, value):
  while (value in a):
    a.remove(value)

def removeAll2(a, value):
  a = copy.copy(a)
  while (value in a):
    a.remove(value)
  return a
```

```
a = [1, 2, 3, 4, 3, 2, 1]
print("a =", a)

removeAll1(a, 2)
print("After removeAll1(a, 2)")
print(“a =", a)

b = removeAll2(a, 3)
print("After b = removeAll2(a, 3)")
print("a =", a)
print("b =", b)
```

REFERENCES

- You may refer to the following links, if necessary
  - https://docs.python.org/3/library/stdtypes.html#typesseq-mutable
  - https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

LIST COMPREHENSIONS

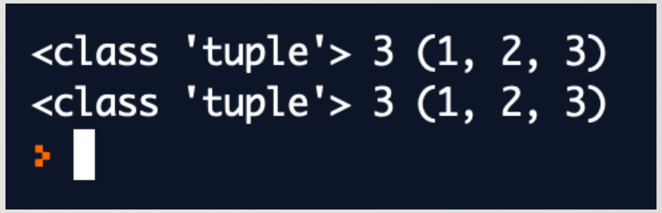
- Help you to write a code concisely

```
a = []
for i in range(10):
  a.append(i)
print("a =", a)

b = [i for i in range(10)]
print("b =", b)

c = [i for i in range(10) if i%2==0]
print("c =", c)
```

TUPLE

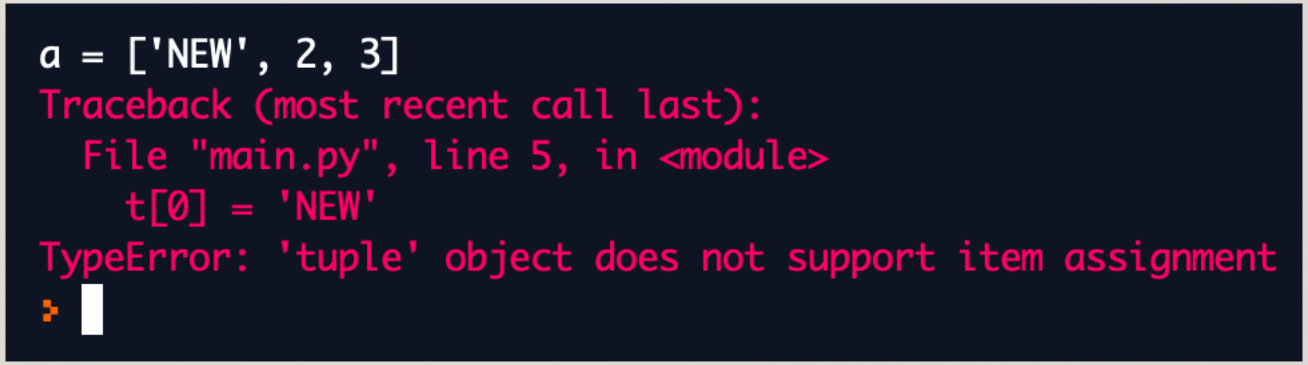
- A new data structure (specialized list, (…) : tuple)
- Tuple is immutable while list is **mutable** (Tuple is **constant**).

```python
t = (1, 2, 3)
print(type(t), len(t), t)

a = [1, 2, 3]
t = tuple(a)
print(type(t), len(t), t)
```



```
<class 'tuple'> 3 (1, 2, 3)
<class 'tuple'> 3 (1, 2, 3)
> 
```

```python
a = [1, 2, 3]
t = tuple(a)

a[0] = 'NEW'
print("a =", a)

t[0] = 'NEW'
print("t =". t)
```



```
a = ['NEW', 2, 3]
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    t[0] = 'NEW'
TypeError: 'tuple' object does not support item assignment
> 
```

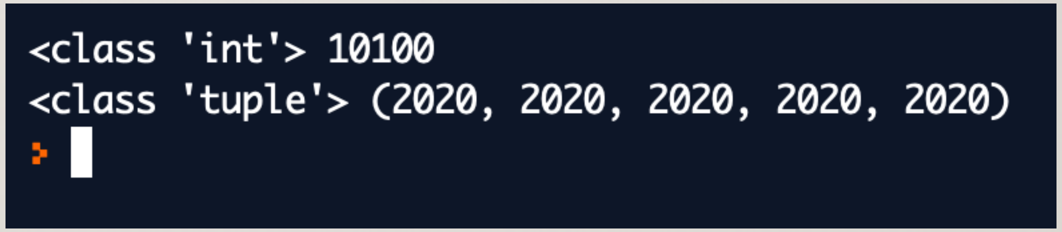
TUPLE

- Parallel tuple assignment (discussed before)

```python
(x, y) = (1, 2)
print("x =", x, "y =", y)

(x, y) = (y, x)
print("x =", x, "y =", y)
```

- Tuple with one element

```python
t = (2020)
print(type(t), t*5)

t = (2020,)
print(type(t), t*5)
```



```
<class 'int'> 10100
<class 'tuple'> (2020, 2020, 2020, 2020, 2020)
```

QUESTION?