ENGG1811 Computing for Engineers

Week 9A: Mutable and immutable data types

You can modify part of a list

- You can modify the elements in a list by assigning new values to them

```
In [11]: x = [11,22,33,43,55]

In [12]: x[3] = 44

In [13]: x
Out[13]: [11, 22, 33, 44, 55]

In [14]: x[2:4] = [37,47]

In [15]: x
Out[15]: [11, 22, 37, 47, 55]
```

String as a sequence of characters

```
In [12]: word = 'silly'

In [13]: word[0]
Out[13]: 's'

In [14]: word[1]
Out[14]: 'i'

In [15]: word[2]
Out[15]: 'l'

In [16]: word[3]
Out[16]: 'l'

In [17]: word[4]
Out[17]: 'y'
```

But you can't modify part of a string

```
In [16]: word = 'silly'

In [17]: word[0] = 'b'
Traceback (most recent call last):

  File "<ipython-input-17-3b299587d77e>", line 1, in <module>
    word[0] = 'b'

TypeError: 'str' object does not support item assignment


In [18]: word = 'billy'
```

← Error

← You can't change part of a string but you can assign an entirely new string

Tuples

- A tuple is a sequence of elements enclosed in ( )
- For example, the numpy where () function returns a tuple, the shape of a numpy function is given in a tuple
- Tuples are in many ways similar to lists
- But you can't modify tuples

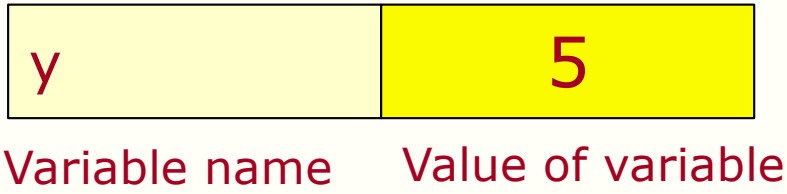
```
In [16]: t = (3,7,21) # A tuple with 3 elements

In [17]: t[1]
Out[17]: 7

In [18]: t[0:2]
Out[18]: (3, 7)

In [19]: t[1] = 10
Traceback (most recent call last):

  File "<ipython-input-19-5a9388635924>", line 1, in <module>
    t[1] = 10

TypeError: 'tuple' object does not support item assignment
```

Mutable and immutable data types

- The data types in Python are divided into 2 kinds
  - Mutable
  - Immutable
- Lists, numpy arrays (and dictionaries) are mutable
  - You can change the individual elements
- Strings are immutable
  - So are int, float, bool, tuples
- Note: dictionaries is a datatype in Python
  - We won't be covering dictionaries in this course

Simplified mental picture on variables [From Week 1]

- Variables are stored in computer memory
- A variable has a name and a value
- A mental picture is:



A program manipulates variables to achieve its goal

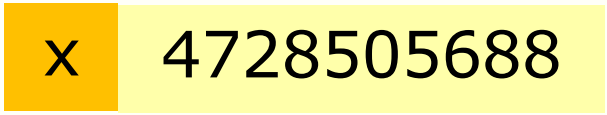
Note: This is a simplified view. We will introduce the more accurate view later in the course.

How Python really stores variables

- In order to understand mutability, we need to understand how Python stores variables

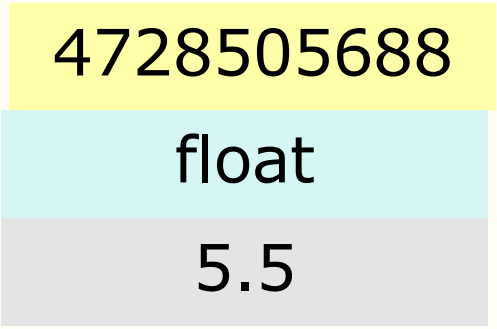
```
In [100]: x = 5.5

In [101]: type(x)
Out[101]: float

In [102]: id(x)
Out[102]: 4728505688
```

Variable x is associated with an identifier



The identifier is associated with the data type and a value.

For a list, a sequence of values



Indirect association

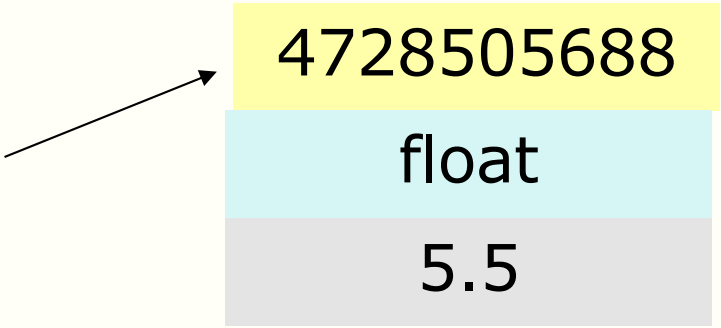
The most important concept that you need to know is that a variable name is associated with its value via an identifier

Variable x is associated with an identifier

x	4728505688

The identifier is associated with the datatype and a value.

For a list, a sequence of values



Copying a mutable type

- We will look at and run the code in mut_1.py

```
14 list1 = [10,11,12,13]
15 list2 = list1
```

```
id of list1 =  4728419656
id of list2 =  4728419656
```

list1	4728419656

list2	4728419656

4728419656
list
10,11,12,13

Note: You will **not** get the same id shown above when you run the program. The essence is whether list1 or list2 have the same or different id

Lessons learnt

- The key lessons learnt from mut_1.py are
  - There are two different ways to copy lists

```
list2 = list1
```

Note: Both variable names are associated with the SAME list

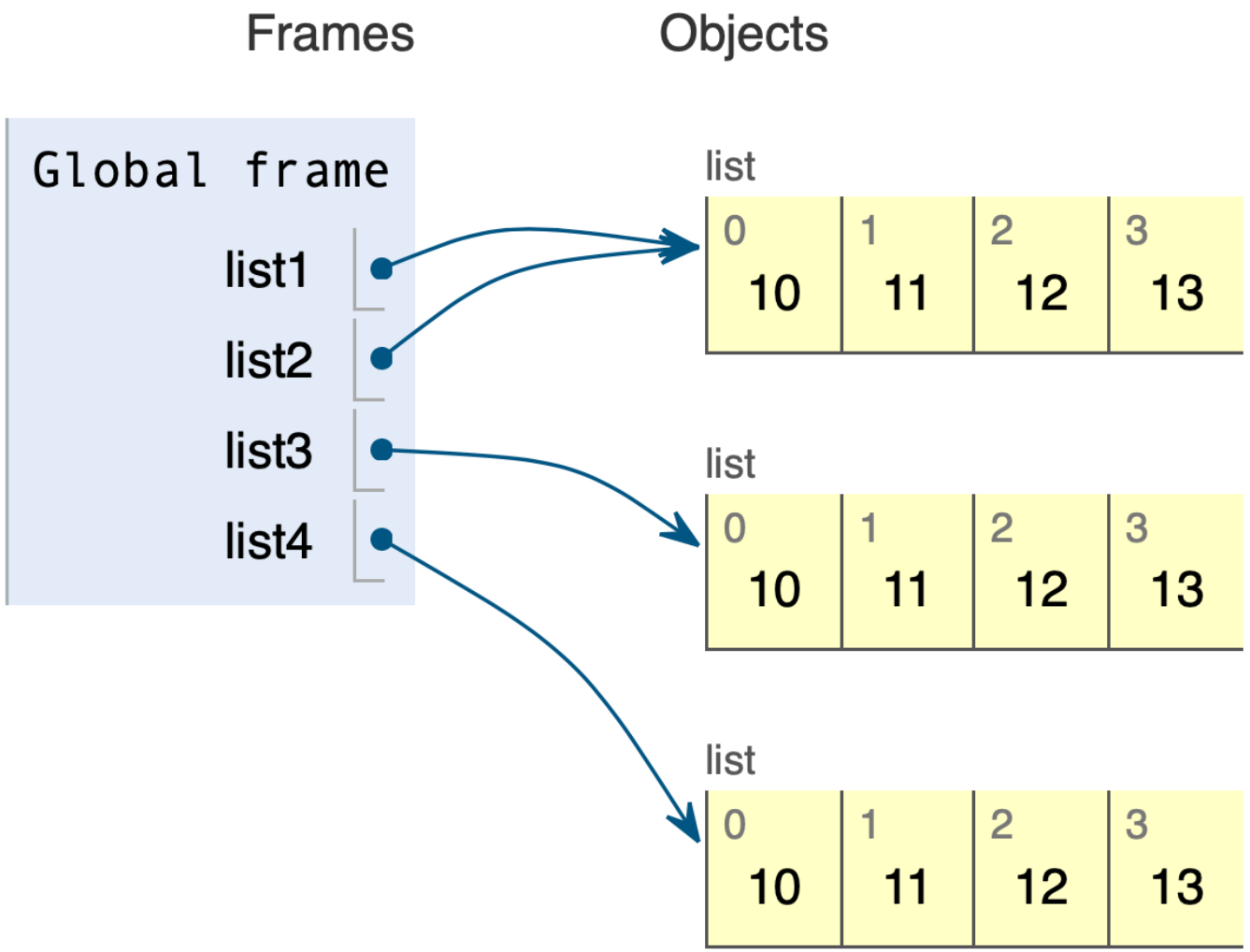
```
list4 = list3[:]
```

Note: The variable names are associated with different list

You can visualise the code on Python tutor.

See the screenshot from Python tutor on the next page.

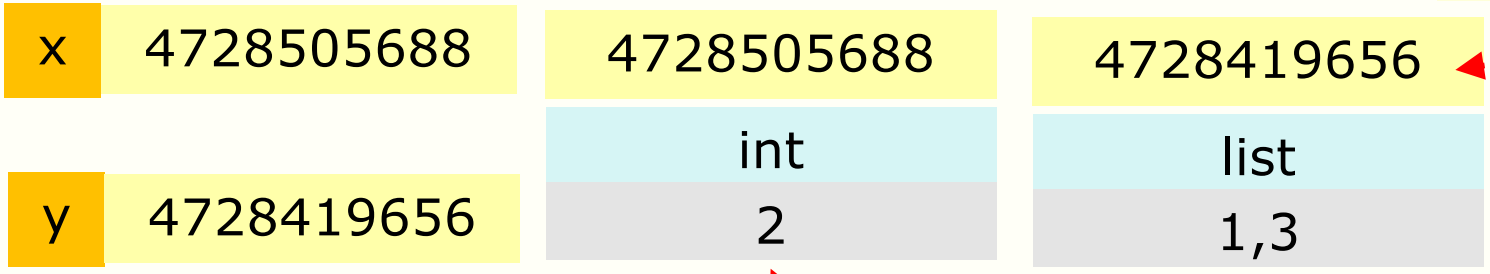
```
1  list1 = [10,11,12,13]
2  list2 = list1
3
4  list3 = [10,11,12,13]
5  list4 = list3[:]
```



Pass by value / reference

- Python function treats its input according to whether it is mutable or immutable
- E.g., consider a function with 2 inputs
  - x is of immutable type
  - y is of mutable type

```
x = 2
y = [1,3]
z = func(x,y)
```



- For an input of **immutable** type, the function will be told its **value**
- E.g. func() will be told that the value of x is 2
- Pass by value

- For an input of **mutable** type, you can choose to tell the function the input's **identifier**
- E.g. func() will be told the identifier of y
- Pass by reference

Modifying list using functions

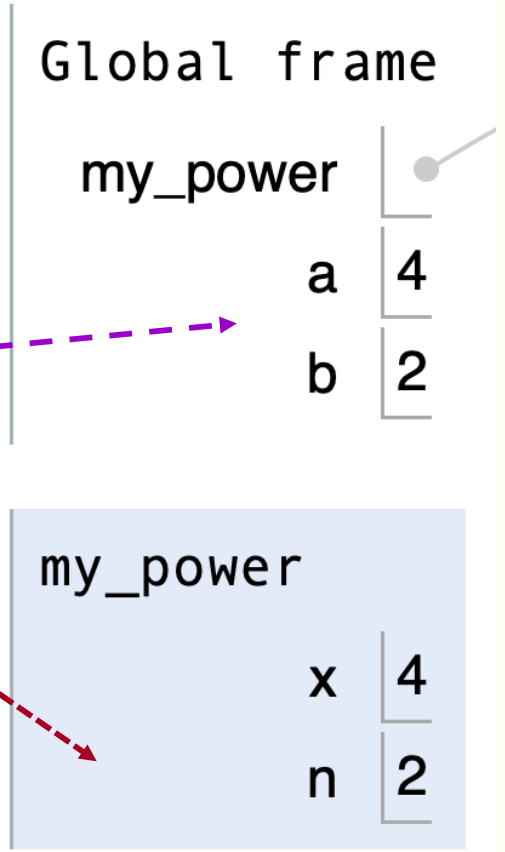
- We say in Week 2 that the scope of the variables in a function is local. This is true for immutable objects.
  - See the next slide
- For mutable data type, you can modify them by using functions
  - This is a consequence of pass by reference
- Let us look at the examples in mut_2.py

Pass by value (immutable type)

- In the example below, the values 4 and 2 are passed to the function
- The function does not modify the variables a and b
- Separate memory spaces for the variables within the function

```
def my_power(x,n):
    y = x ** n
    return y


a = 4; b = 2
z = my_power(a, b)

print('y = ' ,y)
print('z = ' ,z)
```

x ← 4
n ← 2



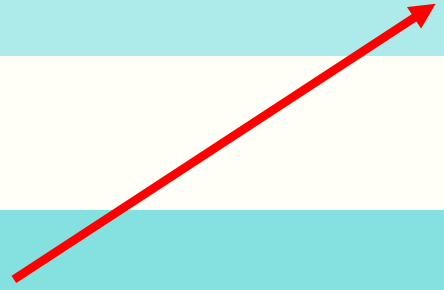
Pass by reference

↓ From mut_2.py

```
def extend(input_list):
    input_list.append(-1)

list0 = [5, 11, 12, 13]
extend(list0)
```

Memory space of the function extend

Input_list	4728419656



Memory space

list0	4728419656

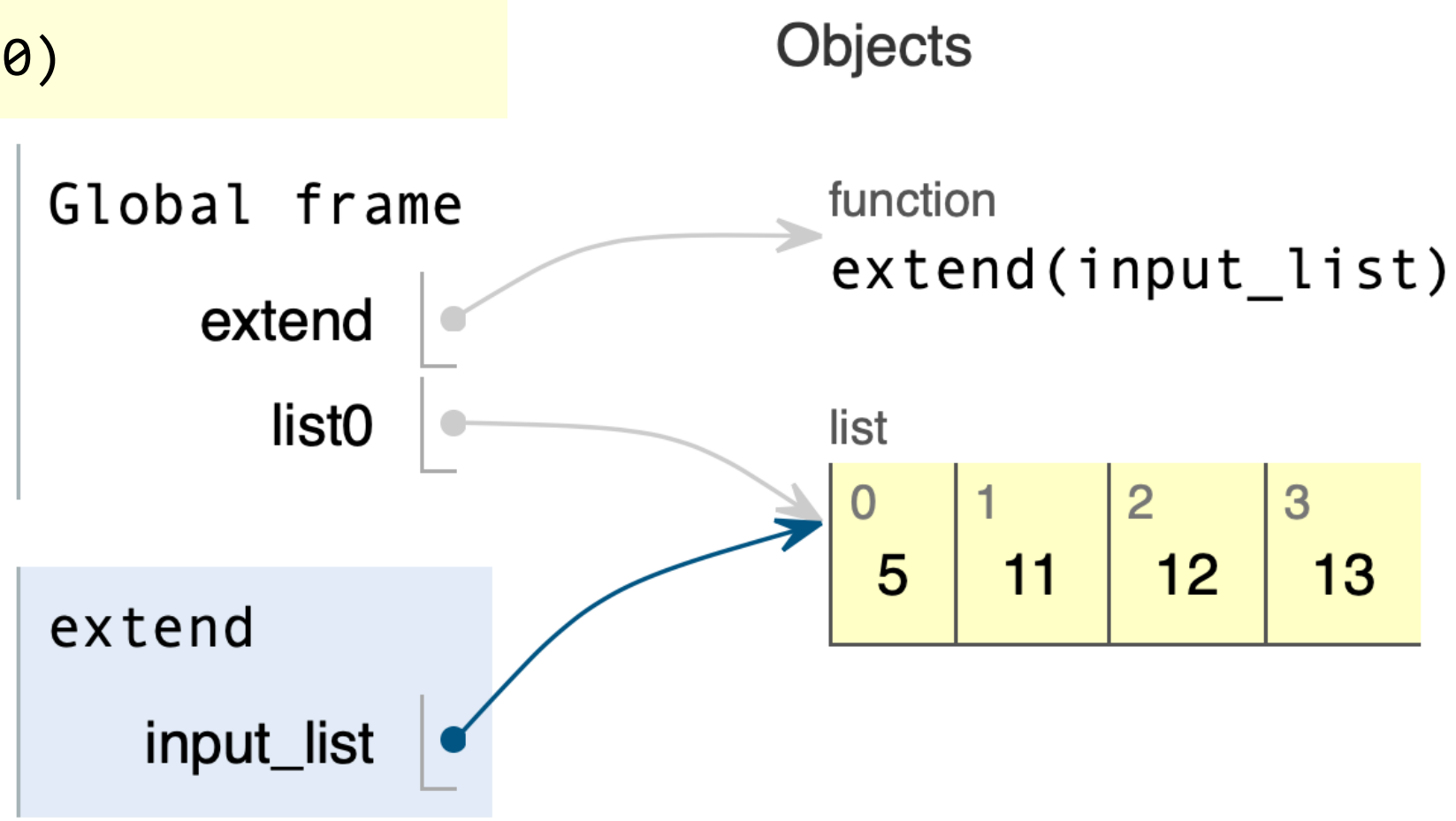
4728419656
list
...

When the function extend is called, this identifier is passed to the function. With the identifier, the function can locate the list. The identifier refers to the list, hence the name pass by reference.

Memory requirement: passing list by reference

```
def extend(input_list):
    input_list.append(-1)

list0 = [5, 11, 12, 13]
extend(list0)
```

← Need memory to store list0 only



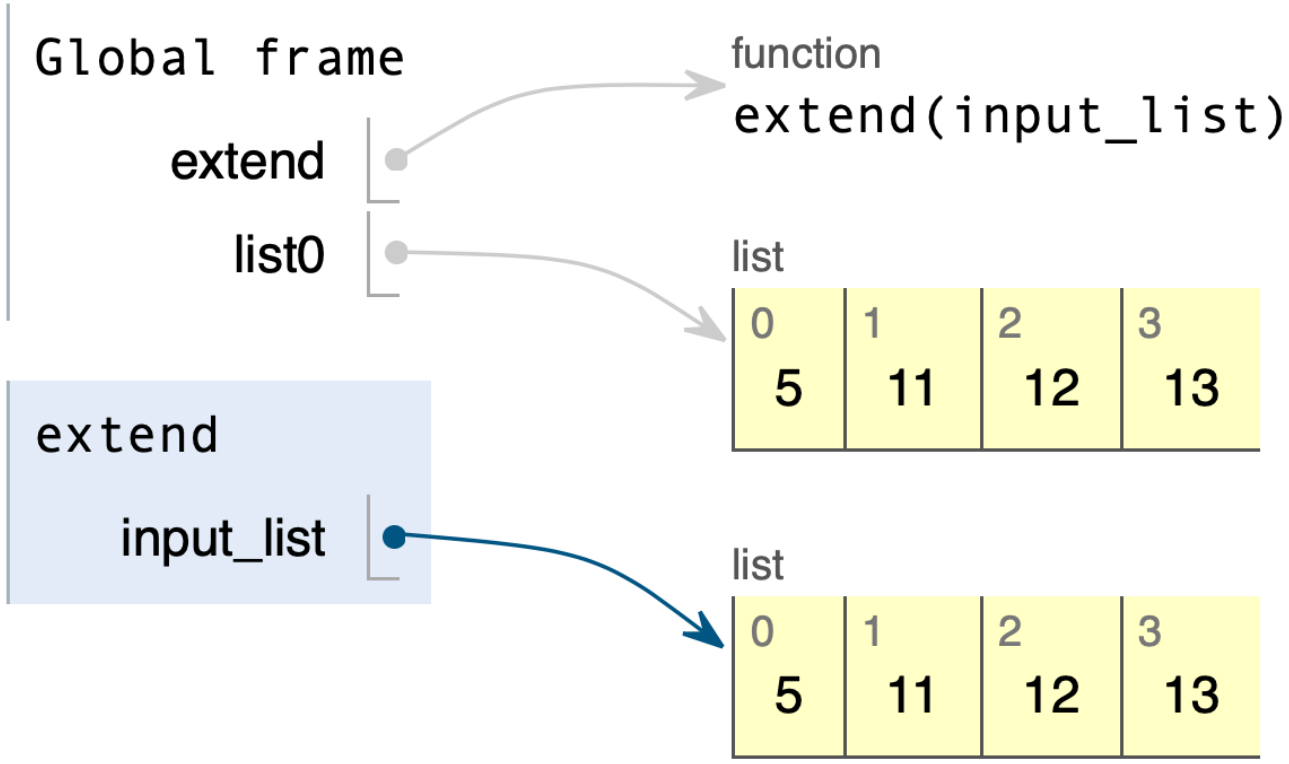
Memory requirement: passing list by value

```
def extend(input_list):
    input_list.append(-1)

list0 = [5, 11, 12, 13]
extend(list0[:])
```

Need memory to store list0 and memory for a copy of list0 in the function.

Double the memory requirement

The list is now passed by value.



Why mutable data types?

- Allow pass by reference
  - Lower memory requirement. Saves time to locate vacant memory and to duplicate the list.
  - Beneficial if the list is long
  - More data is collected than in the past, so large data sets become more prevalent

numpy arrays

- numpy arrays are mutable

- If you want to copy the contents of an array into another without associating them, you need to use the numpy function copy()
  - See mut_3.py

Beware

- Need to remember that lists and numpy arrays (in general mutable types) can be modified by your functions
- Sometimes you may find that your lists or arrays have been changed (mysteriously) even though you have not worked on them directly.
- This can be because you have modified them unknowingly in some functions
- The example on the next page shows you how you can identify whether your function is the culprit

- Check your function to see whether a mutable input appears on the left-hand side of an assignment operation
  - list1 will be modified by the function
- From mut_2.py →

- To avoid this problem:
  - Use pass by value →
  - Make an independent copy of the array ↓

```
#%% Example 2 (Part 1) Pass by reference
def double(input_list):
    for k in range(0,len(input_list)):
        input_list[k] = 2 * input_list[k]
    return input_list

list1 = [5, 11, 12, 13]
print('list1 = ',list1)
list2 = double(list1)
print('list1 = ',list1)
print('list2 = ',list2)
```

```
# %% Example 2 (Part 2) Pass by value
def double(input_list):
    for k in range(0,len(input_list)):
        input_list[k] = 2 * input_list[k]
    return input_list

list3 = [5, 11, 12, 13]
print('list3 = ',list3)
list4 = double(list3[:])
```

```
# %% Example 2 (Part 3) Make an independent copy
def double(input_list):
    output_list = input_list[:]
    for k in range(0,len(output_list)):
        output_list[k] = 2 * output_list[k]
    return output_list
```

Summary

- Immutable: int, float, bool, str, tuple
- Mutable: list, numpy array

- Different ways to copy mutable types

- Pass by value, pass by reference

- Passing by reference for list, numpy arrays
  - Beware that the function can modify the list/array
  - Memory requirement