

# Python: brief introduction

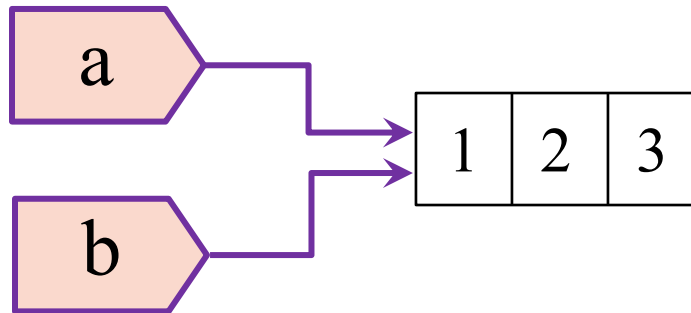
# 1.1. Types

```
#clear|  
a=2  
b=3.0  
c=a+b  
d=2*a
```

- A) c is float, d is float
- B) c is float, d is int
- C) c is int, d is int
- D) c is int, d is float

# 1.2. Names and values

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



The list 

1	2	3
---	---	---

 is an object, and both *names*

a
---

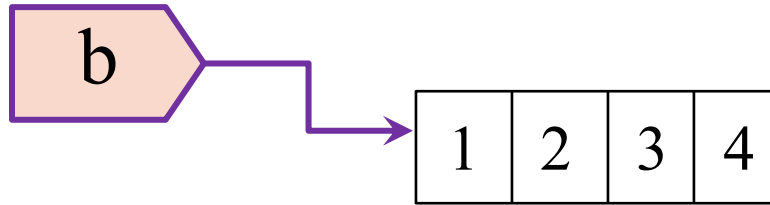
 and 

b
---

 are bounded to the same list (*values*)

# Modifying an object

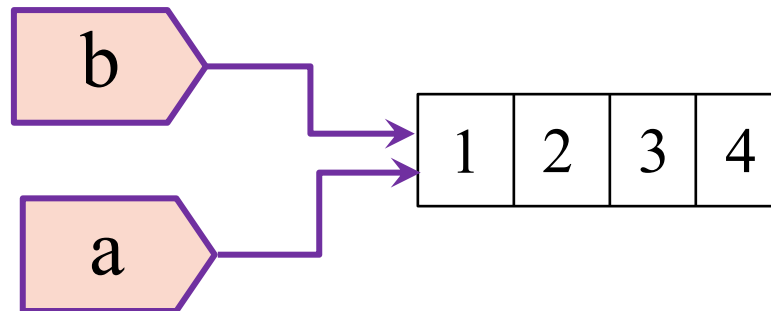
```
a = [1, 2, 3]
b = a
b.append(4)
```



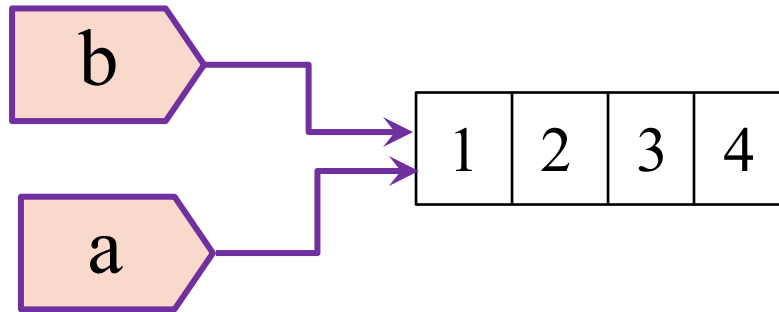
`b.append(4)` modifies the object list [1,2,3]

What happens to the name “a”?

Because “a” and “b” are bounded to the same location, they will have the same values once the list is modified



# Get the “id” for an object



```
#clear  
print(id(a), id(b))
```

```
2053127830536 2053127830536
```

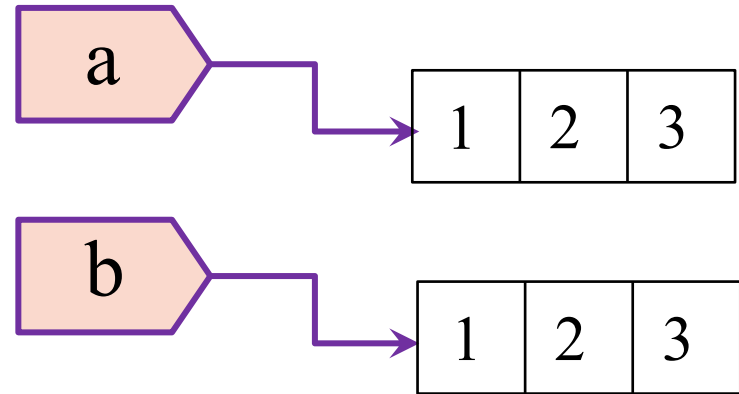
Since “a” and “b” are bounded to the same object, then they have the same “id”

```
#clear  
a is b
```

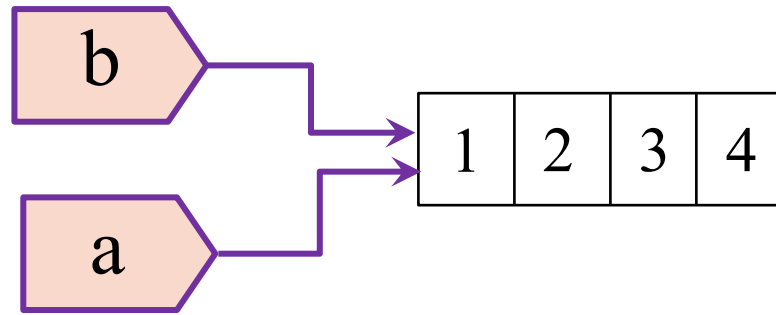
Check if both names have the same “id”

# In summary ...

```
a = [1,2,3]
b = [1,2,3]
print("IS", a is b)
print("EQUAL", a == b)
```



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



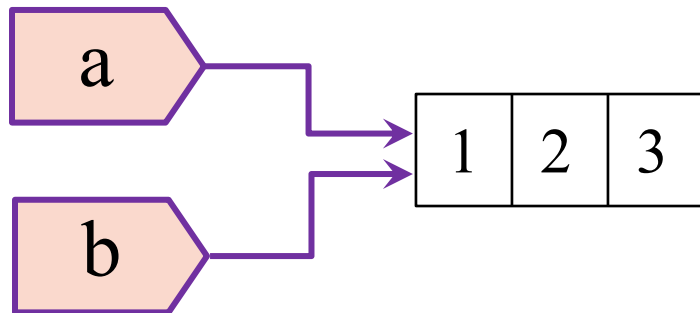
# Mutable and immutable types

**Mutable objects:** can be changed after they are created (e.g. lists, dictionaries)

**Immutable objects:** cannot be changed after they are created (e.g. tuples, strings, floats)

**Mutable object: List**

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



```
a = a + [4]
print(b)
print(a)
a is b
```

```
a += [4]
print(b)
print(a)
a is b
```

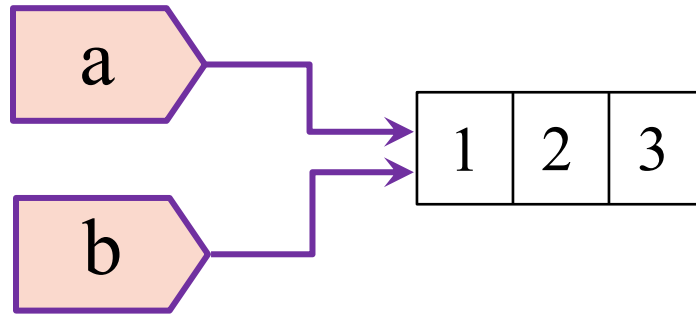
Do you get the same results when running these two pieces of code?

A) YES

B) NO

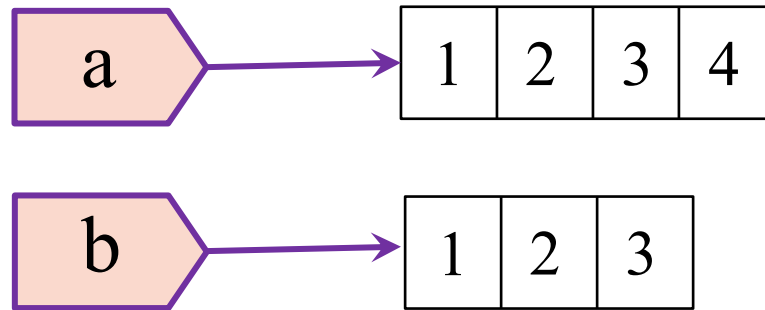
## Mutable object: List

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



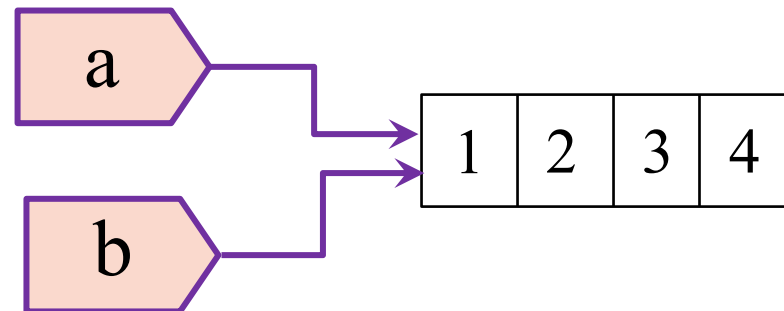
“a” gets reassigned to a new object, “b” is still bounded to the initial object.

```
a = a + [4]
print(b)
print(a)
a is b
```



The object list is modified, however, “a” and “b” remain bounded to the object.

```
a += [4]
print(b)
print(a)
a is b
```





## 1.2. Names and values

Which of the following code snippets

A)

```
a = ['hello', 'goodbye']  
b = 'hey'  
a.append(b)  
c = a + [b]
```

C)

```
a = ['hello', 'goodbye']  
b = 'hey'  
c = a + [b]  
a.append(b)
```

B)

```
a = ['hello', 'goodbye']  
b = 'hey'  
c = a + [b]  
a += b
```

Results in

```
print(a==c)
```

True

# 1.3. Advanced Names

```
fruit = 'apple'  
  
lunch = []  
lunch.append(fruit)  
  
dinner = lunch  
dinner.append('fish')  
  
fruit = 'pear'  
  
meals = [fruit, lunch, dinner]  
print(meals)
```

# 1.3. Naming advanced

What is the correct output for the following code snippet?

```
John = 'computer_science'  
Tim = John  
Tim += ', math'  
Anna = ['electrical']  
Julie = Anna  
Julie += ['physics']  
print(John, Anna)
```

## Choice\*

- A)  computer\_science, math ['electrical', 'physics']
- B)  computer\_science, math ['electrical']
- C)  computer\_science ['electrical', 'physics']
- D)  computer\_science ['electrical']

# 1.4 Indexing

```
a = [0,1,2,3,4,5,6,7,8,9]
```

$a[i:j:k]$       $i$  – starting index  
                   $j$  – stopping index (not included)  
                   $k$  – step

```
a = [0,1,2,3,4,5,6,7,8,9]
```

```
a[1::2][::-1]
```

What is the output for the command line above?

- A) [1,3,5,7,9]
- B) [1,3]
- C) [3,1]
- D) [9,7]
- E) [9,7,5,3,1]

# 1.5 Control Flow

```
#clear  
mylist = []  
  
for i in range(50):  
    if i % 7 == 0:  
        mylist.append(i**2)
```

```
mylist  
[0, 49, 196, 441, 784, 1225, 1764, 2401]
```

```
#clear  
mylist = [i**2 for i in range(50) if i % 7 == 0]  
print(mylist)
```

```
[0, 49, 196, 441, 784, 1225, 1764, 2401]
```

# 1.6 Functions

```
def add_minor(person):  
    person.append('math')  
  
def switch_majors(person):  
    person = ['physics']  
    person.append('economics')  
  
John = ['computer_science']  
Tim = John  
add_minor(Tim)  
switch_majors(John)  
print(John, Tim)
```

## Choice\*

- A)  ['computer\_science', 'economics'], ['computer\_science', 'economics']
- B)  ['physics', 'economics'], ['computer\_science']
- C)  ['physics', 'economics'], ['physics', 'economics']
- D)  ['computer\_science', 'math'], ['computer\_science', 'math']
- E)  ['physics', 'economics'], ['computer\_science', 'math']

# 1.7 Objects

```
#clear
class test:
    def __init__(self):
        self.variable = 'Old'
        self.Change(self.variable)
    def Change(self, var):
        var = 'New'
obj=test()
print(obj.variable)
```

- A) Error message, because the function Change can't be called in the `__init__` function
- B) 'Old'
- C) 'New'

A)

```
a = [3,4]
b = [6,7]

def do_stuff(a,b):
    return( a.append(5), b.append(8) )

do_stuff(a,b)
```

B)

```
a = 3
b = 5

def do_stuff(a,b):
    a += 1
    b += 2

do_stuff(a,b)
```

C)

```
a = [3,4]
b = [6,7]

def do_stuff(a,b):
    a += [5]
    b += [8]

do_stuff(a,b)
```

Which code snippet does not modify the variables?



## 2.2 Numpy Indexing

```
a = np.array([[1, 4, 9], [2, 8, 18]])
```

## 2.3 Broadcasting

```
a = np.arange(9).reshape(3, 3)
print(a.shape)
print(a)
```

```
b = np.arange(4, 4+9).reshape(3, 3)
print(b.shape)
print(b)
```

```
a = np.arange(9).reshape(3, 3)
print(a.shape)
print(a)
```

```
b = np.arange(3)
print(b.shape)
print(b)
```

## 2.3 Broadcasting

Given A and B numpy arrays such that:

A.shape is (5,4)

B.shape is (1,4)

What is the shape of  $A + B$ ?

A)(1,4)

B)(5,1,4)

C)(5,4)

D)Not a valid operation