

# List Versus Tuples

# How do we use lists and tuples differently

- Before this, let's revisit “functions”

# Let's Write Our Own Function!

Define  
(keyword)

Function name

Input  
(Argument)

```
def square(x):  
    return x * x
```

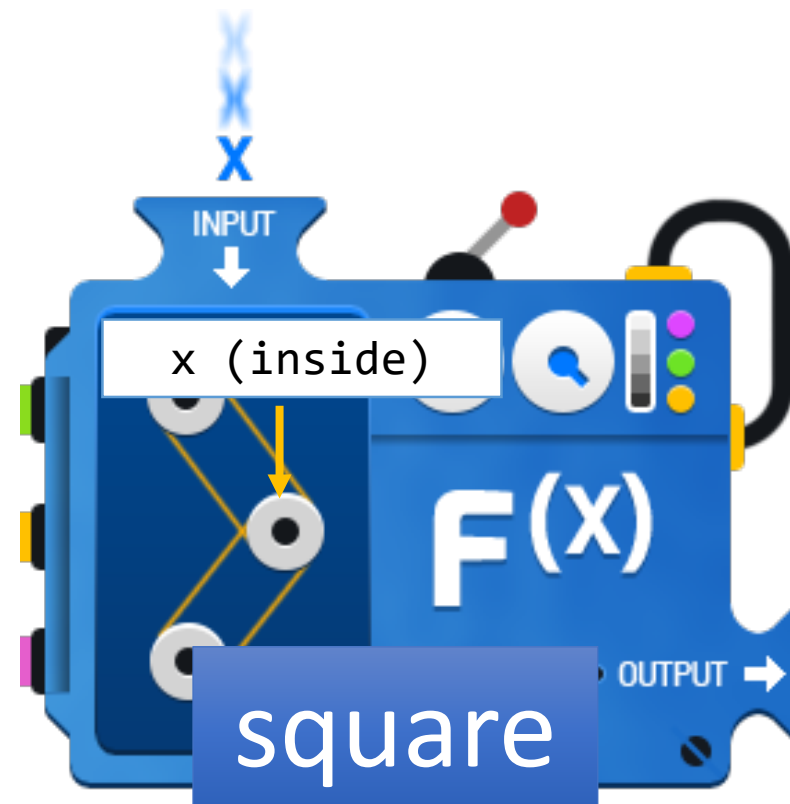
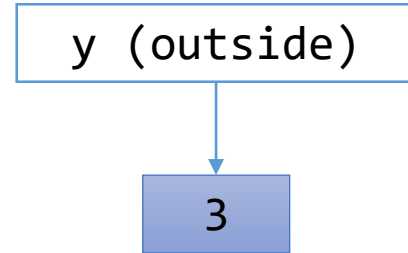
Indentation

Output

# For example

- “square” is a function

```
>>> y = 3  
>>> square(3)  
9
```

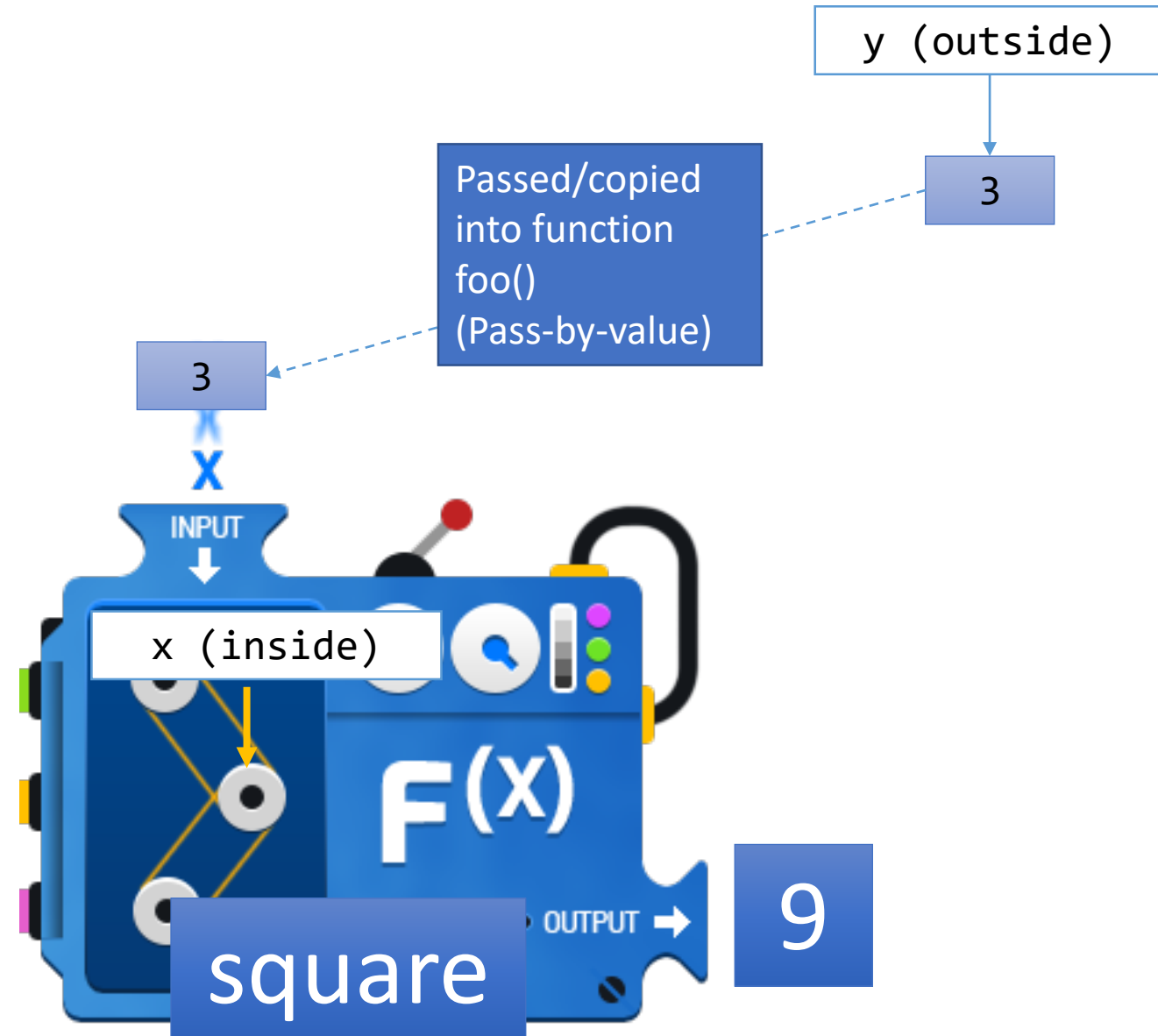


# More Precisely

- “square” is a function

```
>>> y = 3
>>> square(3)
9
```

- Because the “3” that enters the function square is a copy of y
  - Meaning, the outside y will **NOT** be changed
  - This is under the assumption of “pass-by-value”



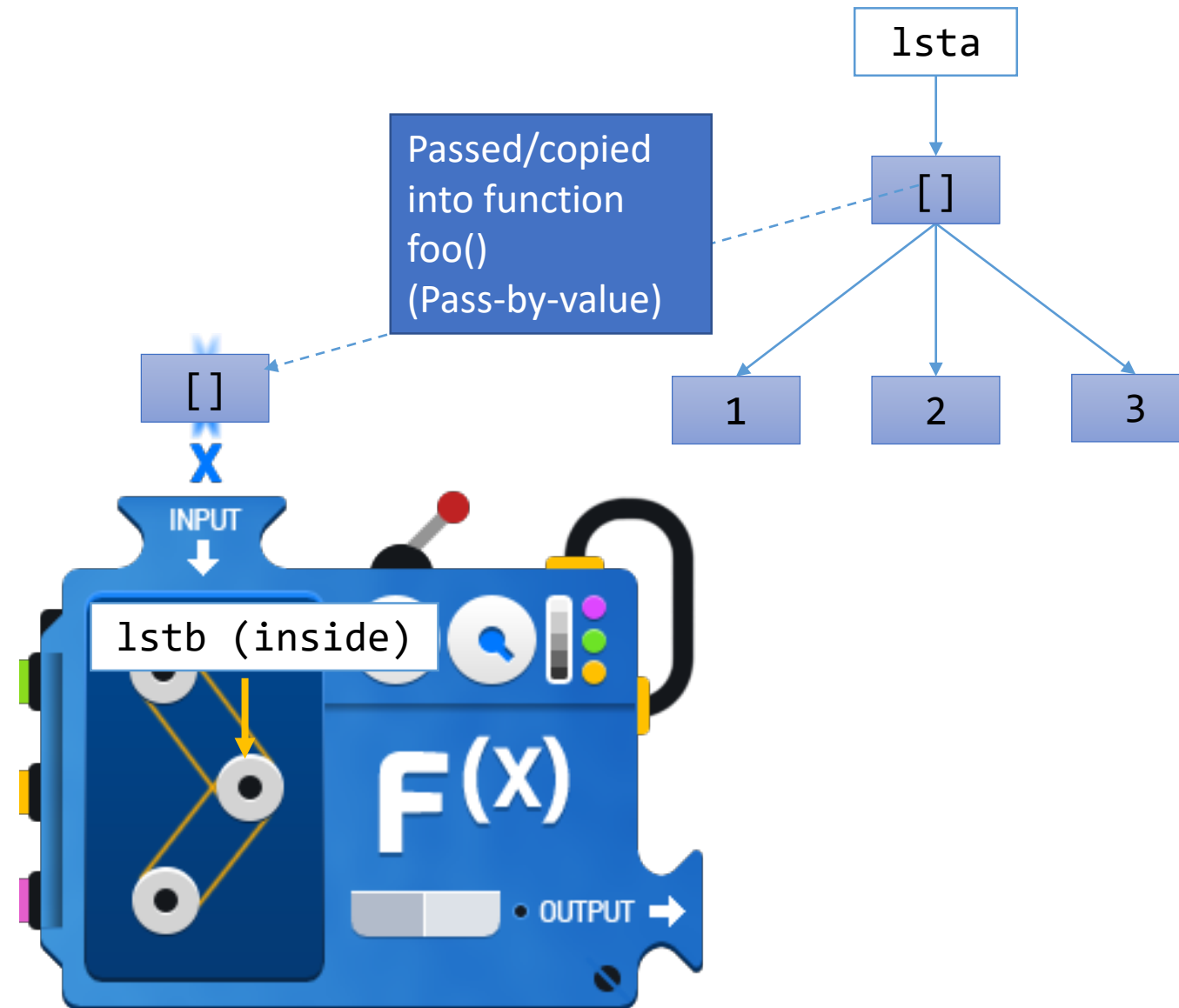
# However,

- If we pass a list into a function

```
lsta = [1,2,3]
```

```
def f(lstb):  
    lstb[0] = 999
```

```
f(lsta)  
print(lsta)
```



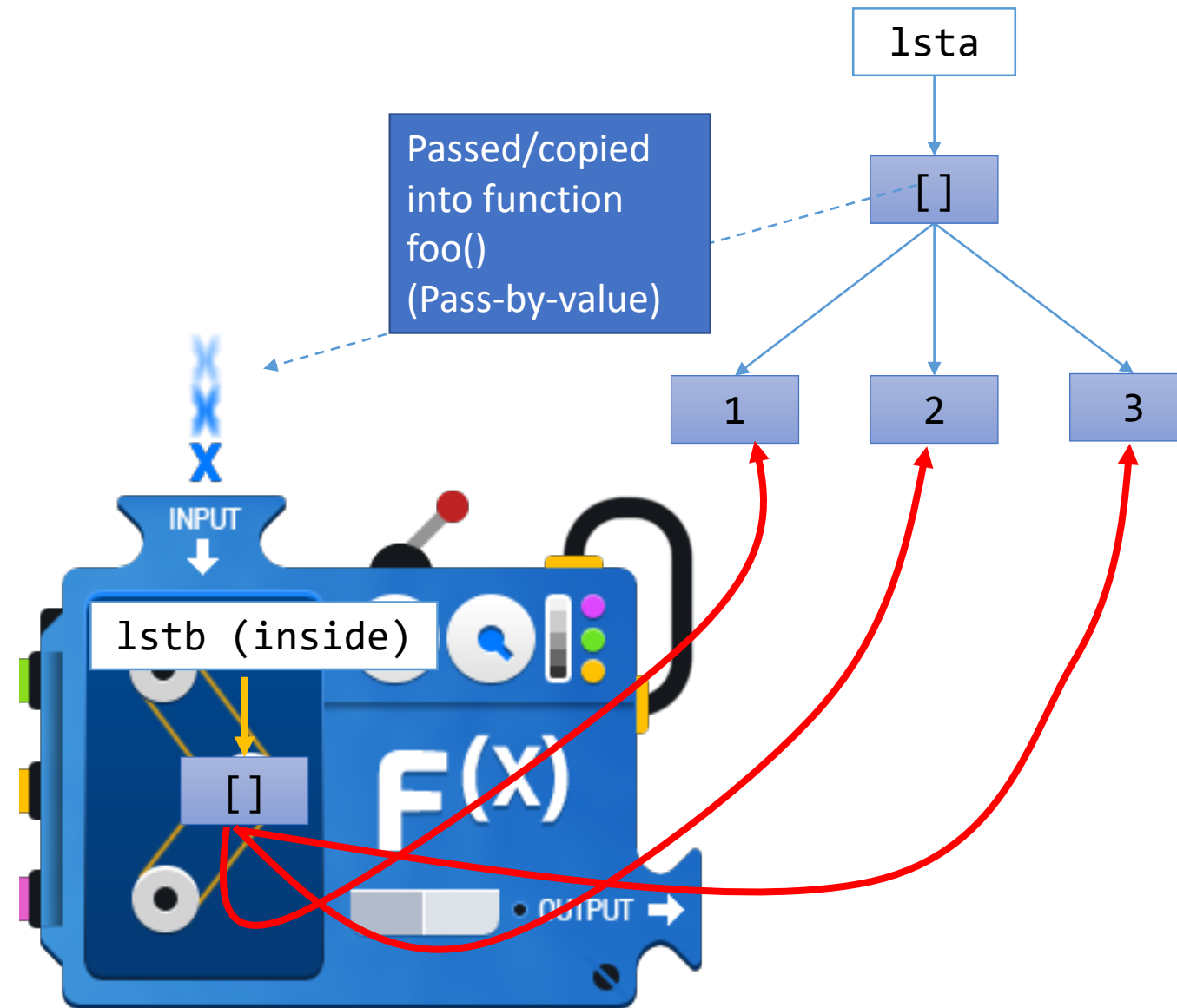
# However,

- If we pass a list into a function

```
lsta = [1,2,3]
```

```
def f(lstb):  
    lstb[0] = 999
```

```
f(lsta)  
print(lsta)
```



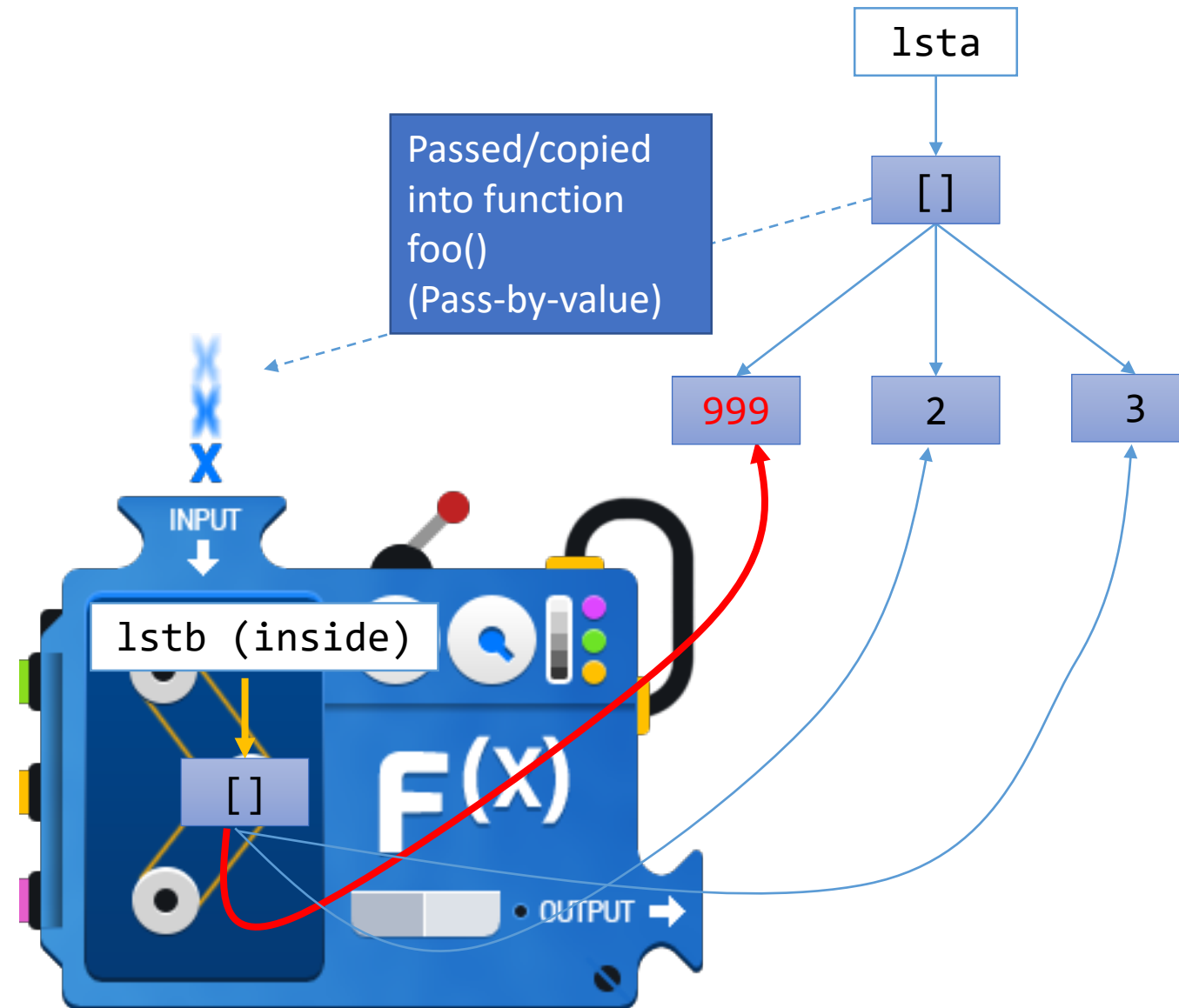
# However,

- If we pass a list into a function

```
lsta = [1,2,3]
```

```
def f(lstb):  
    lstb[0] = 999
```

```
f(lsta)  
print(lsta)
```





# You may say there are TWO types of functions

- Type Pass-by-value:
  - Functions that will **NOT** modify the inputs (aka arguments or parameters)

```
>>> l = [3, 4, 5, 6, 1, 9]
>>> sorted(l)
[1, 3, 4, 5, 6, 9]
>>> l
[3, 4, 5, 6, 1, 9]
```

- Type Pass-by-reference (or pass-by-pointers)
  - Functions that **WILL** modify the inputs (aka arguments or parameters)

```
>>> l
[3, 4, 5, 6, 1, 9]
>>> l.sort()
>>> l
[1, 3, 4, 5, 6, 9]
>>>
```

# To Modify a Sequence, which one is better?

- Say, we just simply want to add one element into a sequence
- Lists:

`[1, 2, 3, 4, 5]`       $\rightarrow$       `[1, 2, 3, 4, 5, 999]`

- Tuples

`(1, 2, 3, 4, 5)`       $\rightarrow$       `(1, 2, 3, 4, 5, 999)`

- If I want to write a function to achieve these, what is the difference?

# Tuples

- Say, `tup = (1, 2, 3, 4, 5)`
- I want to pass it into a function `modifyTup()` and produce a longer tuple
- However, I cannot modify the original tuple

# You may say there are TWO types of functions

- Type Pass-by-value:

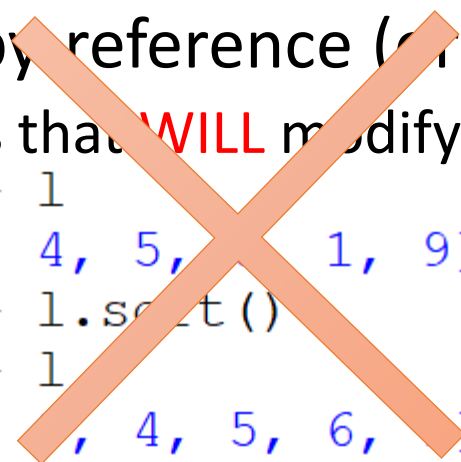
- Functions that will **NOT** modify the inputs (aka arguments or parameters)

```
>>> l = [3, 4, 5, 6, 1, 9]
>>> sorted(l)
[1, 3, 4, 5, 6, 9]
>>> l
[3, 4, 5, 6, 1, 9]
```

- Type Pass-by-reference (or pass-by-pointers)

- Functions that **WILL** modify the inputs (aka arguments or parameters)

```
>>> l
[3, 4, 5, 1, 9]
>>> l.sort()
>>> l
[1, 4, 5, 6, 9]
>>>
```



# Tuples

- Say, `tup = (1, 2, 3, 4, 5)`
- I want to pass it into a function `modifyTup()` and produce a longer tuple
- However, I cannot modify the original tuple
- The only way I can do is to return a longer tuple by `modifyTup()`

```
def modifyTup(t):  
    return t + (999,)
```

- And use it like this

```
>>> tup = (1, 2, 3, 4, 5)  
>>> tup = modifyTup(tup)  
>>> tup  
(1, 2, 3, 4, 5, 999)
```

# Compare These Two

- Tuples

```
def modifyTup(t):  
    return t + (999,)
```

```
>>> tup = (1, 2, 3, 4, 5)  
>>> tup = modifyTup(tup)  
>>> tup  
(1, 2, 3, 4, 5, 999)
```

- Integers

```
def modifyInt(x):  
    return x + 1
```

```
>>> x = 4  
>>> x = modifyInt(x)  
>>> x  
5
```

# Lists

- Say, `lst = [1, 2, 3, 4, 5]`
- I want to pass it into a function `modifyLst()` and produce a longer **list**
- ~~However, I cannot modify the original list~~

# You may say there are TWO types of functions

- Type Pass-by-value:

- Functions that will **NOT** modify the inputs (aka arguments or parameters)

```
>>> l = [3, 4, 5, 6, 1, 9]
>>> sorted(l)
[1, 3, 4, 5, 6, 9]
>>> l
[3, 4, 5, 6, 1, 9]
```



- Type Pass-by-reference (or pass-by-pointers)

- Functions that **WILL** modify the inputs (aka arguments or parameters)

```
>>> l
[3, 4, 5, 6, 1, 9]
>>> l.sort()
>>> l
[1, 3, 4, 5, 6, 9]
>>>
```





# Lists

- Say, `lst = [1, 2, 3, 4, 5]`
- I want to pass it into a function `modifyLst()` and produce a longer **list**
- ~~However, I cannot modify the original list~~
- ~~The only way~~ I can do is to return a longer **list** by `modifyLst()`

```
def modifyLst(l):  
    return l + [999]
```

- And use it like this

```
>>> lst = [1, 2, 3, 4, 5]  
>>> lst = modifyLst(lst)  
>>> lst  
[1, 2, 3, 4, 5, 999]
```

# Lists Version 2

- Say, `lst = [1, 2, 3, 4, 5]`
- I want to pass it into a function `modifyLst()` and produce a longer **list**
- However, how about this:

```
def modifyLstV2(l):  
    return l.append(999)
```

- And use the function like this

```
>>> lst = [1, 2, 3, 4, 5]  
>>> modifyLstV2(lst)  
>>> lst  
[1, 2, 3, 4, 5, 999]
```

# What is the difference between the two versions?

```
def modifyLst(l):  
    return l + [999]
```

```
>>> lst = [1,2,3,4,5]  
>>> lst = modifyLst(lst)  
>>> lst  
[1, 2, 3, 4, 5, 999]
```

- This return the value to copy to “lst”

```
def modifyLstV2(l):  
    return l.append(999)
```

```
>>> lst = [1,2,3,4,5]  
>>> modifyLstV2(lst)  
>>> lst  
[1, 2, 3, 4, 5, 999]
```

- This modify the original input “lst”

# Two Types of Functions

- Type Pass-by-value:

- Functions that will **NOT** modify the inputs (aka arguments or parameters)

```
>>> l = [3, 4, 5, 6, 1, 9]
>>> sorted(l)
[1, 3, 4, 5, 6, 9]
>>> l
[3, 4, 5, 6, 1, 9]
```

- Type Pass-by-reference (or pass-by-pointers)

- Functions that **WILL** modify the inputs (aka arguments or parameters)

```
>>> l
[3, 4, 5, 6, 1, 9]
>>> l.sort()
>>> l
[1, 3, 4, 5, 6, 9]
>>>
```

# What is the difference between the Three Versions?

- Let's simply try to apply the “modify” function 100000 times to see how fast they run?
- `time()` is from the package “time”
  - It will return the number of seconds passed since “epoch” time
    - 1<sup>st</sup> Jan 1970 00:00:00

```
ts = time()
tup = ()
for i in range(100000):
    tup = modifyTup(tup)
print(f'Tuple Version: {round(time()-ts,3)}s')
```

```
ts = time()
lst = []
for i in range(100000):
    tup = modifyLst(lst)
print(f'List Version: {round(time()-ts,3)}s')
```


```
ts = time()
lst = []
for i in range(100000):
    tup = modifyLstV2(lst)
print(f'List Version 2: {round(time()-ts,3)}s')
```

# What is the difference between the Three Versions?

- Let's simply try to apply the “modify” function 100000 times to see how fast they run?
- Result:

Tuple Version: 13.03s  
List Version: 0.019s  
List Version 2: 0.017s

More than  
double



- 200000 times:

Tuple Version: 66.499s  
List Version: 0.039s  
List Version 2: 0.037s

```
ts = time()
tup = ()
for i in range(100000):
    tup = modifyTup(tup)
print(f'Tuple Version: {round(time()-ts,3)}s')
```

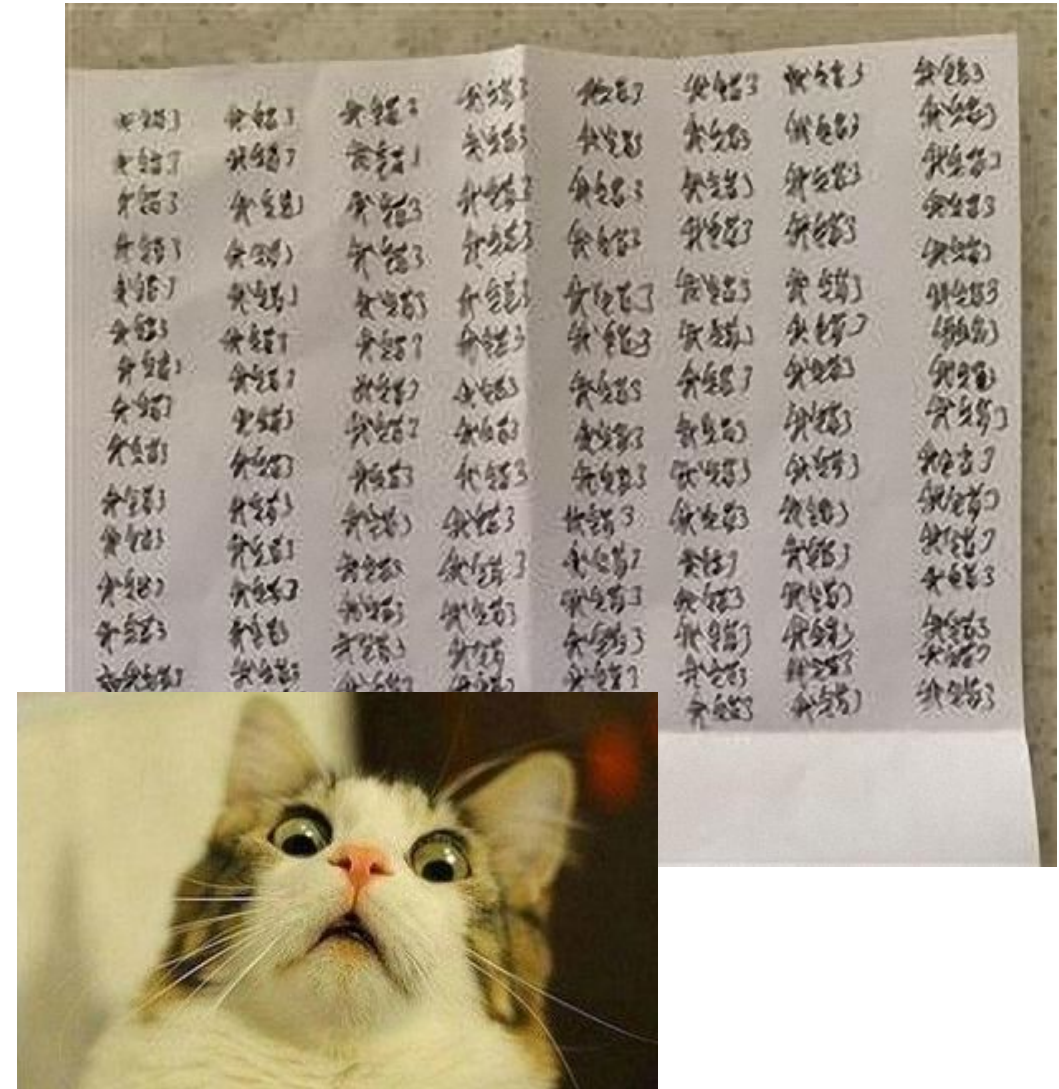
```
ts = time()
lst = []
for i in range(100000):
    tup = modifyLstV1(lst)
print(f'List Version 1: {round(time()-ts,3)}s')
```

```
ts = time()
lst = []
for i in range(100000):
    tup = modifyLstV2(lst)
print(f'List Version 2: {round(time()-ts,3)}s')
```

WHY SO SERIOUS?

# Why is the Tuple Version so Slow?

- When I was young
- When I was naughty in school, teachers punished me with “copying”
  - A punishment of copying some phrase (in Chinese) many times
- One time, when I submitted my copying of **50** times
  - Teacher said it should be **100** times
- Should I...
  - Copy 100 times from scratch? Or
  - Copy 50 times and add my existing ones?



# To Copy or Not to Copy?

- Copy 100 times all over from scratch? Or
  - Copy 50 times and add my existing ones?
- 
- What is the difference if my evil teacher keeps increasing 50 times whenever I submit?
    - Copy all over, or
    - Add to existing one?



# Why is the Tuple Version so Slow?

- This line copy the **whole** tuples to tup again
- So if you want to do it 100000 times
  - The first time copy 6 items
  - The second copy time 7 items
  - 8,9,10,..... 1000005
- Number of times the sum of all

```
tup = ()
for i in range(100000):
    tup = modifyTup(tup)
print(f'Tuple Version: {round(time()-ts,3)}s')
```

```
def modifyTup(t):
    return t + (999,)
```

```
>>> tup = (1,2,3,4,5)
>>> tup = modifyTup(tup)
>>> tup
(1, 2, 3, 4, 5, 999)
```

# Comparing

- Tuples:
  - $6+7+8+\dots+100005$
  - $= 5,000,550,000$

# Why is the Tuple Version so Slow?

- However, for **list**, you just add one item to the end
  - No re-copying of the previous existing items in the list
- Therefore
  - The first time 1
  - The second time 1
  - 1,1,1,1,.....1
- In totally only 100000 times

```
def modifyLstV2(l):  
    return l.append(999)
```

```
lst = []  
for i in range(100000):  
    tup = modifyLstV2(lst)
```

# Comparing

- Tuples:
  - $6+7+8+\dots+100005$
  - $= 5,000,550,000$
- Lists:
  - $1+1+1+1+1+1+\dots+1+1$
  - $= 100000$
- Lists win!

# To Copy or Not to Copy?

- Copy 100 times all over from scratch? Or
  - Copy 50 times and add my existing ones?
- 
- What is the difference if my evil teacher keeps increasing 50 times whenever I submit?
    - ~~Copy all over, or~~
    - Add to existing one!

# Back to my school copying...



# Conclusion

## Tuple Version



Preserve input



Must return result



Slower

## List Version



May or may not preserve input

- May modify input



May or may not need to return result



Faster