Let's Write Python
[2]

By the way, I am so sorry ပါဗျာ။ မနေ့ ကလဲလုပ်စရာလေးတွေရှိလို့ အပြင်ထွက်နေရတာ ညဘက်မှပြန်ရောက်တယ်ဗျ။ ဒီနေ့ လဲ Ubuntu လေးပြန်သုံးဦးမယ်ဆိုပြီး 14.10 လေးတင်ပြီး သွင်းစရာရှိတာတွေသွင်းနေတာနဲ့ တိုင်ပတ်နေလို့ ညဘက်မှပဲရေးပြီးတင်ပေးလိုက်ပါတယ်။ (ဘယ်ပဲရောက်ရောက် CoC ကတော့တက်ဖြစ်တယ်ဗျ, MSF(Priv8) မှာတက်တက်ယွတာလေ 😜)

ပထမပိုင်းမှာတုံးကျွန်တော်ပြောခဲ့တဲ့ Python Interpreter မှာ 0 နဲ့ စတာတွေ 010 ဆိုဘာလို့ 8 လို့ တယ်ပေါ့။ ရေးလိုက်တာနဲ့ ပြန်ဖော်ပြတာကိုကြည့်ပြီးသဘောပေါက်ပြီးသားသူလဲသိမှာပါ။ Python Interpreter မှာက 0 နဲ့ စလိုက်ရင် octal or base 8 အဖြစ်နဲ့ interpreted လုပ်ပါတယ်။

ဒါဆိုအပိုင်း၂လေးဆက်လိုက်ရအောင်ဗျာ...

Lists & Tuples

Lists နဲ့ Tuples နဲ့ မှာအဓိကကွဲပြားတာကတော့ List က changes ရှိနိုင်တယ်ဒါပေသိ Tuples တွေကတော့ဒီလိုမဟုတ်ပါဘူး ပြောရရင် fix ဖြစ်တယ်ပေါ့။ ဒါကလဲလိုအပ်တဲ့နေရာမှာသင့်တော်တဲ့ကောင်ကိုရွေးသုံးသွားနိုင်တာပေါ့လေ။
Lists တွေကို variable အသစ်ထည့်ထည့်သွားတဲ့အခြေအနေမျိုးတွေမှာအသုံးချနိုင်ပြီး Tuples တွေကိုတော့ changes မပြုလုပ်နိုင်အောင်ပြုလုပ်ထားသင့်တဲ့ sequence တွေမျိုးမှာအသုံးချပါတယ်။ သူ့ နေရာနဲ့ သူအသုံးတည့်တာကြီးပဲပေါ့။
Tuples အကြောင်းကိုတော့နောက်မှ Detailed သွားမယ်ဗျာ။ ဟုတ်ပြီဒါဆိုကျွန်တော်တို့ Lists ဆိုတာကိုအသေးစိတ်လေ့လာလိုက်ကြရအောင်ဗျာ။

2. Lists

2.1 Lists are actually a sequence.

String ကဲ့သို့ ပါပဲ List ဆိုတာကလဲတကယ်တော့ Sequence of values တွေဖြစ်ပါတယ်။ နောက် List တစ်ခုမှာဆို data type ကကြိုက်တာရပါတယ်။ List တစ်ခုထဲမှာရှိတဲ့ values တွေကို elements ဒါမှမဟုတ် items လို့ ခေါ်ကြပါတယ်။ List တွေသည် Square Bracket ဆိုတဲ့ [] ကိုအသုံးပြုပါတယ်။

Code:

```
>>> [1, 2, 3, 4, 5]
>>> ['I', 'Love', 'MSF']
```

ပထမ List ကတော့ Integer 5 ခုပါ။ ဒုတိယကောင်ကတော့ string 3 ခုပါဂင်တဲ့ List တစ်ခုပါ။ ပြောဖို့ တစ်ခုရှိတယ်ဗျ ဒီဥပမာတွေသာကျွန်တော်ကာ same data type နဲ့ ရေးပြသွားတာနော်။ List မှာအပေါ်ကပြောခဲ့သလိုပဲ any data types ပါ။ အောက်ပါဥပမာမှာ string, list, float နဲ့ တခြား List တစ်ခုပါထပ်ထည့်ပြပါမယ်။

Code:

```
>>> ['Hello', 1, 1.5, [11, 'World']]
```

Lists တွေထဲမှာအပေါ်ကလိုနောက်ထပ် Lists တစ်ခုထပ်ထည့်လိုက်တာမျိုးတွေကို nested လို့ ခေါ်ပါတယ်။ နောက် ဘာမှမထည့်ထားတဲ့ [] ေတွလဲရေးရတာမျိုးတွေရှိတတ်ပါတယ်။ ဘယ်လိုပြောရမလဲဗျာ စာသောက်ဆိုင်ကစားပွဲထိုးတစ်ယောက်ကလာစားတဲ့သူမှာတဲ့ဟာကိုမှတ်ရတာကို program ရေးလိုက်ရင်ပေါ့။ အရင်ဆုံးဘာမှမရေးထားတဲ့စာရွက်အလွတ်တစ်ရွက် (Program ထဲမှာတော့ဘောပင်ပေါ့ထည့်မရေးရဘူးပေါ့ကွယ် 😜) အဲ့လိုစာရွက်အလွတ်ကို Empty List ဆိုတဲ့ [] ပေါ့။ နောက် customer ကမှာလိုက်တာတွေကိုစာရွက်မှာရေးပြီးမှတ်သလိုပဲ ခုနက List ထဲကိုလဲထည့်မှတ်သွားတာပေါ့ Python အရပြောရရင် List ထဲကို append လုပ်တယ်ပေါ့။ ဒီလိုပါ။

Code:

```
paper = [] # Creating Empty List
ask = raw_input('What would you like to order: ') # Asking for the input
paper.append(ask) # Adding User Value into Empty List
print paper # Print out the results from the list
```

append() ဆိုတာကတော့ List ထဲကို Value ထည့်ခွင်အသုံးပြုတဲ့ function name လေးတစ်ခုပေါ့။

2.2 Wow, Lists are mutable

ဟုတ်ပါတယ်။ Lists တွေက mutable ဖြစ်ပါတယ်။ Interpreter မှာအောက်ကလို List လေးတစ်ခုရေးလိုက်ပြီထားဗျာ။

Code:

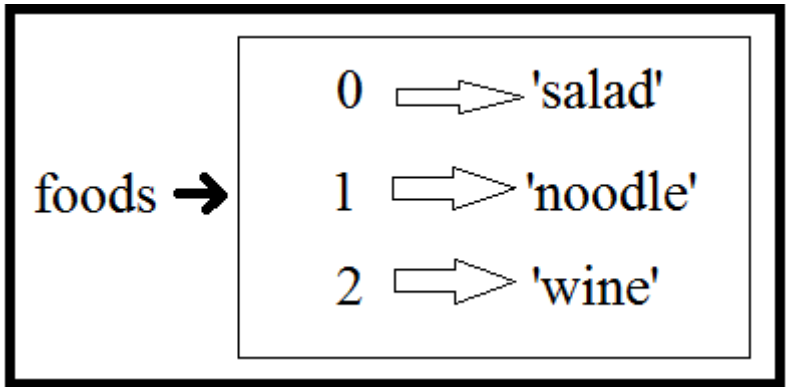
```
>>> foods = ['salad', 'noodle', 'wine']
>>> numbers = [1, 2, 3]
```

ပထမ List လေးကိုအရင်ပြောမယ်။ အဲ့ဒီ foods ဆိုတဲ့ Lists ထဲကဟာတွေအကုန်မပြပဲတစ်ခုပဲပြခွင်တယ်ဆိုဘယ်လိုရေးမလဲ? ကျွန်တော်က wine ဆိုတဲ့ကောင်ကိုခေါ်ပြပါမယ်။

Code:

```
>>> foods[2]
'wine'
```

ဘာလဲဟ wine က 3th place မှာလေ.... တတိယကောင်မဟုတ်ဘူးလားးးး ဒါပေသိ သတိထားရမှာက count ကို 1 ကနေမစပါဘူး 0 ကစပါတယ်။ အောက်ကပုံလေးကိုကြည့်လိုက်ပါ။

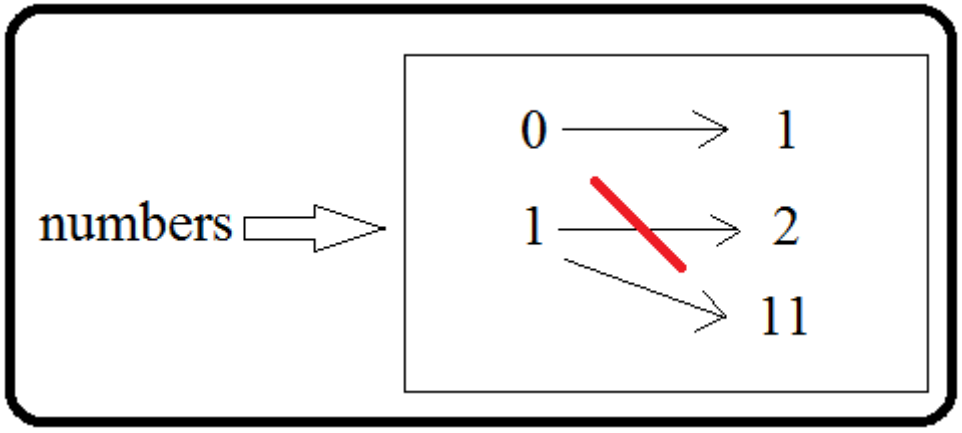


ရှင်းတယ်မလား ဒါဆိုနောက်တစ်ကောင် numbers = [1, 2, 3]. Lists တွေက mutable ဖြစ်တယ်လို့ ပြောခဲ့တယ်နော် ဒီငနဲကြီးကပြောတယ်နော်၊ ဒါဆို 2 ဆိုတဲ့နံပတ်ကိုမကြိုက်ဘူးကွာ 11 ပြောင်းမယ်.....။ ရပါတယ်ဗျာ ဒီလိုလေးပြောင်းပါမယ်။

Code:

```
>>> numbers[1] = 11
>>> print numbers
[1, 11, 3]
```

ဘာမှတောင်မှတ်စရာမလိုဘူးရယ် ခုနကလိုတစ်ခုချင်းခေါ်သလိုခေါ်တာလေးရေး နောက်က equal sign နဲ့ ပြောင်းမယ့် value ရေးလိုက်ရုံလေးတင်.....



List ဆိုတာကို indices နဲ့ elements တွေလို့ မှတ်ထားလိုက်ရင်အဆင်ပြေပါတယ်။ အဲ့ဒီ indices နဲ့ elements တွေရဲ့ကြားကဆက်စပ်မှုကို mapping လို့ ခေါ်ပါတယ်။ element တစ်ခုစီကို index တစ်ခုစီနဲ့ ဆက်စပ်နေပါတယ်။

List indices အလုပ်လုပ်ပုံကတော့

• ဘယ် interger expression ကို index အနေနဲ့ အသုံးချနိုင်ပါတယ်။
• မရှိတဲ့ value တစ်ခုကိုရေးဖို့ ပဲဖြစ်ဖြစ် ဖော်ပြဖို့ ပဲဖြစ်ဖြစ်ပြုလုပ်လိုက်ပါက IndexError ဖော်ပြမှာဖြစ်ပါတယ်။
• Index တွေကို negative values (such as: -1, -2) တွေနဲ့ ဖော်ပြရင် နကို list values တွေကို backward ပြန်ရည်တွက်ပါတယ်။ ဥပမာပေးရမယ်ဆို

Code:

```
>>> print numbers[-1]
3
```

in ဆိုတဲ့ keyword ကိုအသုံးချပြီး lists ထဲမှာ element ရှိမရှိဆိုတာကို True or False နဲ့ ဖော်ပြခိုင်းနိုင်ပါတယ်။ ခုနက foods ဆိုတဲ့ list နဲ့ ဖော်ပြပါမယ်။

Code:

```
>>> foods = ['salad', 'noodle', 'wine']
>>> 'wine' in foods
True
>>> 'cocktail' in foods
False
```

2.3 Traversing a list

List တစ်ခုထဲက elements တွေကို pass over လုပ်ပြီးဖော်ပြသွားတာပြောရရင် traversing လုပ်တဲ့အခါကျရင်အသင့်တော်ဆုံးက for ကိုအသုံးပြုပြီး looping လုပ်သွားတာပါ။

Code:

```
>>> for things in foods:
…    print things
```

(PS: တစ်ခုပြောမယ်နော် print ဆိုတဲ့အကြောင်းရေးတဲ့အခါမှာ white space အနေနဲ့ tab ခုန်လို့ ရပေမဲ့မခုန်ပါနဲ့ ။ space-bar ကိုပဲ ၄ချက်ပုတ်ပြီးအသုံးပြုပေးပါ။ ဒါလေးကိုလဲမှတ်ထားပါ... ဘာ Script ပဲရေးရေးအဲ့လိုအသုံးပြုတဲ့အကျင့်လေးလုပ်ပေးပါ။ http://www.python.org မှာ သွားဖတ်လို့ ရပါတယ်။ Rule လေးတစ်ခုပါပဲ ဒါဟာလဲ။)

ဒါဆို Lists အကြောင်းပဲပြန်ဆက်မယ်ဗျာ.... အခုလို functions လေးကတော့ တခြားမဟုတ်ပါဘူး foods ဆိုတဲ့ထဲက elements တွေကိုတစ်ခုချင်းဖော်ပြပေးသွားတာပါ။ နောက်ဥပမာထဲမှာ things လို့ သုံးပေမယ့်ညီကိုတွေကြီးက်တာရေးလို့ ရပါတယ်။ တစ်ခုပါပဲအောက်မှပြန်ခေါ်တာနဲ့တော့တူပစေပေါ့။
ဒါဆိုနောက်ကောင်လေးကိုထပ်ပြောမယ်။ အပေါ်မှာ numbers = [1,2,3] လို့ ရေးထားခဲ့သေးတယ်နော်။ ဟုတ်ပြီ အထဲက numbers တွေအကုန်လုံးကို ၂နဲ့ မြောက်မယ်ဆိုရင်ကောာာာ..... ကိုယ့်ဘာသာကို calculator နဲ့ တွက်ပြီးတစ်ခုချင်း numbers[index] = calculated number

ဒါမျိုးတော့ရတာပေါ့လေ။ ဒါပေသိဒီလိုကြီးလုပ်တာက အခုတော့ဘာမှမဖြစ်ပေမယ့် တကယ်တန်းကိုယ်ကရေးတော့မယ်ဆိုအခဲလိုကြီးလုပ်ပြီးအချိန်ကုန်ခံမလို့ လား.... ဒီတော့နောက်ဥပမာလေးတစ်ခုရေးမယ်ဗျာ။

Code:

```
for i in range(len(numbers)):
    numbers[i] *= 2
```

ဒါဆိုမူလရှိတဲ့ numbers ထဲက elements တွေဖြစ်တဲ့ 1, 2, 3 ကဆိုရင် ၂နဲ့ မြောက်လိုက်တဲ့ 2, 4, 6 ဖြစ်သွားမှာပါ။ ဘယ်လိုဖြစ်သွားလဲဆိုတာကိုရှင်းပါမယ်။ ပထမဆုံးရှင်းချင်တာက numbers[i] *= 2 ပါ။
ဒါကကွန်တော်အတိုခေါက်ရေးလိုက်တာပါ။ numbers[i] *= 2 ဆိုတာက numbers[i] = numbers[i] * 2 နဲ့ တူတူပါပဲ။ ရေးရတာတိုသွားတယ်ပေါ့ဗျာ။ အခုလိုအရှည်ကိုဖတ်လိုက်ရင်တော့သဘောပေါက်ပြီထင်ပါတယ်။ ဒါဆိုနောက်တစ်ခု for ဆိုတဲ့စာကြောင်းက len() ဆိုတဲ့ကောင်... သူအလုပ်လုပ်တာကတော့ဘာမှမဟုတ်ပါဘူး parenthesis ထဲက value ရဲ့အရှည်ကိုတွက်ချက်ပါတယ်။ ဒါကိုပြောတာပါ။

Code:

```
>>> len('MSF')
3
```

တစ်ခုပဲ သူက space နေရာကိုလဲ 1 ယူပါတယ်။

```
>>> len('I Love MSF')
10
```

8 မဟုတ်ပါဘူး string ကြားထဲမှာ space ၂နေရာပါတယ် အဲ့ဒီ၂ခုကိုလဲထည့်တွက်ပါတယ်။ နောက် lists တွေထည့်ရင်မှတ်ဖို့ က

```
>>> len(numbers)
3
```

ဟုတ်ပါတယ်။ 3 လို့ ပဲပြမှာပါ။ numbers ထဲက element တစ်ခုကိုမှ 1 ယူတာပါ။ elements က ၃ ခုရှိတဲ့အတွက် ၃ ယူလိုက်တာပါ။
နောက်တစ်ခုပြောချင်တာက range()။ ဒီ function ကတော့ arithmetic ဖြစ်တဲ့ list တစ်ခုပြုလုပ်ပါတယ်။ အပေါ်ကလိုပဲ for loop ပတ်တဲ့နေရာတွေမှာသုံးကြတာများပါတယ်။ range(start,stop[,step]) ဒီလိုလေးပါ။ အထဲက argument က integer ဖြစ်ရပါမယ်။ Interpreter မှာဒီလိုလေးရိုက်ကြည့်ပါ။

Code:

```
>>> range(2)
[0, 1]
```

0 ကစမယ်ပေါ့။ အပေါ်မှာပြခဲ့တဲ့ထဲက step ကို omit လုပ်မယ်။ ပြောရရင်ချန်ခဲ့လို့ ရပါတယ်။ သူက optional argument ပါပဲ။ အကယ်လို့ ချန်ခဲ့လိုက်မယ်အပေါ်ကလို range(2) ဆိုရင် step ကို 1 နဲ့ သွားမှာပါ။ နောက် start ကိုချန်ခဲ့ရင် 0 ကနေစတွက်မှာဖြစ်ပါတယ်။ stop ကတော့ပါရမှာပေါ့လေ။ ဘာ arguments မှမပါရင် Error တက်မှာပါ။

Code:

```
>>> range(1,13,2)
[1, 3, 5, 7, 9, 11]
```

သဘောပေါက်မယ်ထင်ပါတယ်။ နောက် start က stop ထက်ကြီးနေပြီး step က positive နဲ့ ဆို empty list ပြန်မှာပါ။

Code:

```
>>> range(0, -10, 2)
[]
```

နောက် step က negative ဆို reverse သွားမှာပါ။

Code:

```
>>> range(0, -10, 2)
[0, -2, -4, -6, -8]
```

အကယ်လို့ step ကို 0 ပေးလိုက်ရင် ValueError တက်မှာဖြစ်ပါတယ်။ range() function ကတော့ဒါပါပဲ။ ဒါဆိုလိုရင်းကိုဆက်မယ်ဗျာ။

2.4 List operations

Strings တွေလိုပါပဲ + က List မှာလဲ concatenate ပြုလုပ်ပါတယ်။

Code:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

နောက် Strings တွေလိုပါပဲ * က repeat လုပ်တာပါပဲ။

Code:

```
>>> [1] * 3
[1, 1, 1]
>>> [1, 2, 3] * 2
[1, 2, 3, 1, 2, 3]
```

2.5 Slicing the List

Code:

```
>>> a = [1, 2, 3, 4, 5, 6]
>>> a[1:3]
[2, 3]
```

အကယ်လို့ List ရဲ့ First argument ကနေစမယ်ဆို 0 ကနေပဲ့ပါလေ။ ဒါပေသိ အဲ့လို First argument ကနေစမယ်ဆို Slicing ရဲ့ : ရှေ့က number ကို omit လုပ်ခဲ့လို့ ရပါတယ်။ နောက်ဆုံးထိဆို : နောက်ကကောင်ကိုချန်ခဲ့လိုက်ပါ။

Code:

```
>>> a[:5] # is equal to a[0:5]
[1, 2, 3, 4, 5]
>>> a[2:] # is equal to a[2:6]
[3, 4, 5, 6]
```

အကယ်လို့ List တစ်ခုလုံးကိုဖော်ပြမယ်ဆို : ရှေ့ ကောနောက်ကောကိုချန်လိုက်လို့ ရပါတယ်။

Code:

```
a[:]
[1, 2, 3, 4, 5, 6]
```

နေအုံးဘာလို့ အဲ့လို ၂ခုလုံးချန်တာကိုသုံးမှာလဲပဲ့ပါ။ ဒီတိုင်းခေါ်လဲအကုန်ပြတာပဲကို? ဒီလိုရှိတယ်ဖျ List တွေက mutable ဖြစ်တယ်လို့ ကျွန်တော်အပေါ်မှာပြောခဲ့သလိုပဲ operations လုပ်ဆောင်တဲ့အခါက မူလ List ကိုမလုပ်စေပဲ အဲ့ List ကိုကူးပြီးလုပ်ကြပါတယ်။ ဒီတော့ ခုနက [:] ကိုအသုံးပြုပါတယ်။ ဒီလိုပါ။

Code:

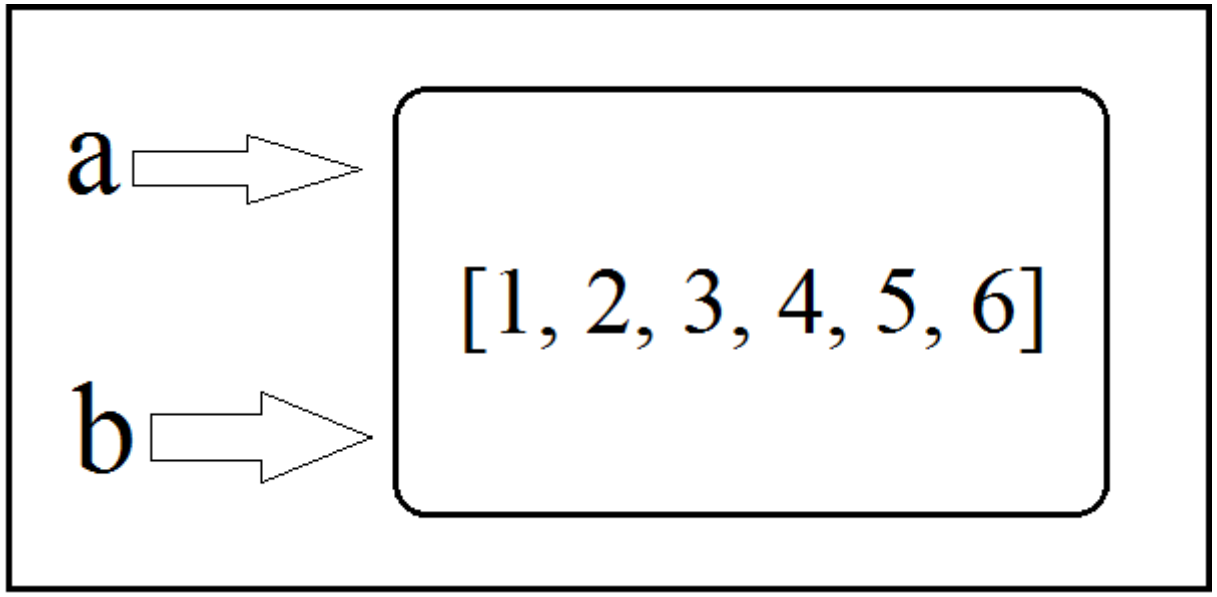
```
>>> b = a[:]
```

ဒီမှာထပ်ပြောစရာရှိသွားပါပြီ။ b = a ဆိုကောမရဘူးလားဆို print ထုတ်ရင်တာတူတူထွက်တာပါ b = a နဲ့ b = a[:] ကမတူပါဘူး။ ဘယ်လိုမတူလဲပြောပြမယ်။
b = a ဆိုတာက b ဆိုတဲ့ကောင်ကို a ရဲ့ List ကိုလှမ်းထောက်ခိုင်းလိုက်တာပါ ဒီတော့ List အသစ်မလုပ်တော့ပဲ a ရဲ့ List ကိုပဲ b ကော a ကအတူတူသုံးကြပါတယ်။ ဒါဆို a ကနေ List ကို edit လုပ်လိုက်ရင် b လဲလိုက်ပြောင်းသွားပါတယ်။

Code:

```
>>> a = [1, 2, 3, 4, 5, 6]
>>> b = a
>>> print a
[1, 2, 3, 4, 5, 6]
>>> print b
[1, 2, 3, 4, 5, 6]
>>> a[0:4] = [11, 22, 33, 44]
>>> print a
[11, 22, 33, 44, 5, 6]
>>> print b
[11, 22, 33, 44, 5, 6]
```

မရှင်းသေးဘူးဆိုအောက်ကပုံကိုကြည့်လိုက်ပါ။



b = a[:] ကကျတော့ a ရဲ့ List ကြီးကို Copy ပြီးအသစ်ထပ်လိုက်လုပ်ပါတယ်။ ပြီးမှ b နဲ့ ပေးညီလိုက်တယ်။ ဒီတော့ a ကို edit ရင် b မှာသက်ရောက်မှုမရှိသလို b ကို edit လဲ a ကိုသက်ရောက်မှာမဟုတ်ပါဘူး။

Code: