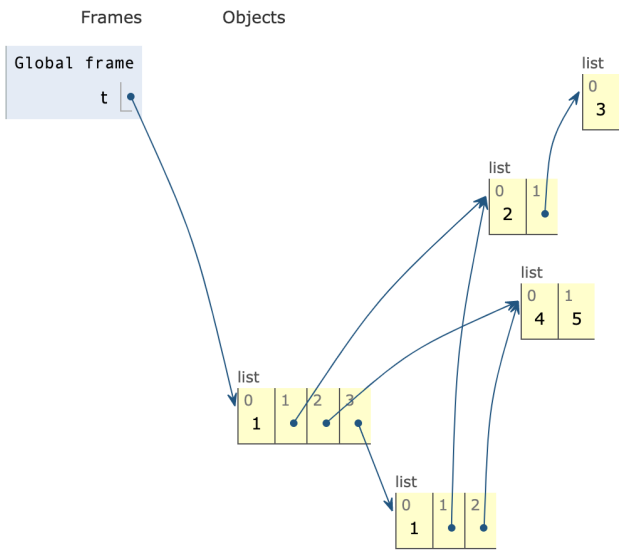
CS61A NOTE5 Mutability

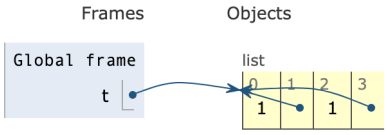
Mutability

a=[1,2,3,4]
a.append([a.pop()])
b=a[2:]
b[1].extend(a[2:1:−1])
b.append(a.pop())
a[2]=b.pop()
a[2][−2]=9
a[2].extend(b)

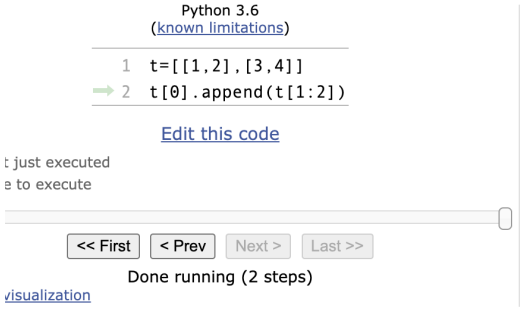
t=[1,[2,[3]],[4,5]]
t.append(t[:])

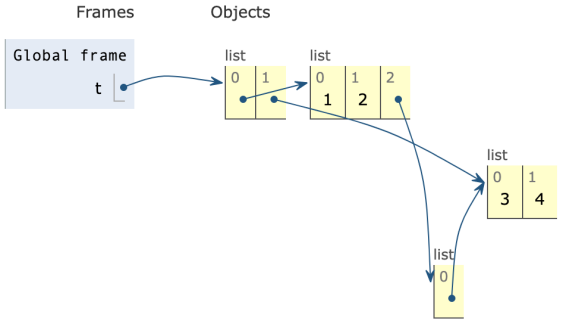


t=[1,2,3]
t[1:3]=[t]
t.extend(t)

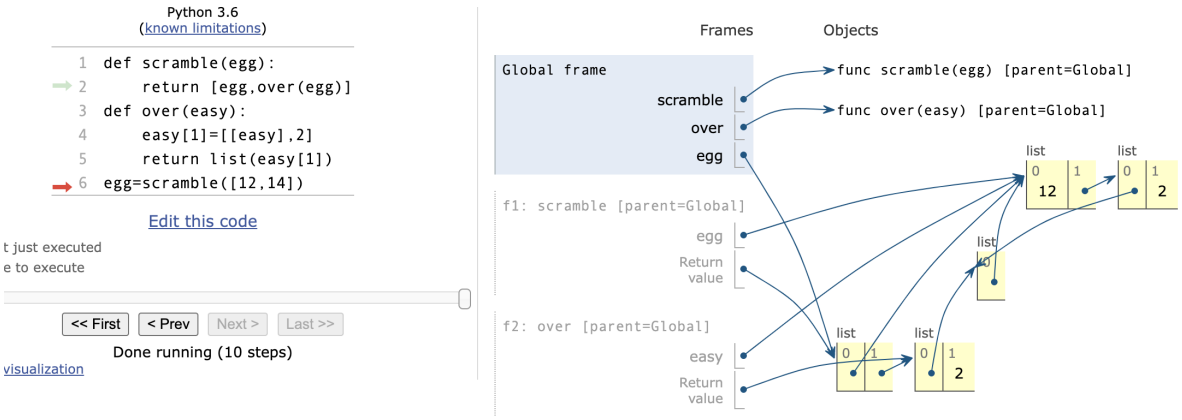


t=[[1,2],[3,4]]
t[0].append(t[1:2])






```
def scramble(egg):
    return [egg,over(egg)]
def over(easy):
    easy[1]=[[easy],2]
    return list(easy[1])
egg=scramble([12,14])
```



ex=[1,2,3]

def foo(x):

bar[x]=x

bar.append(len(bar))

return bar

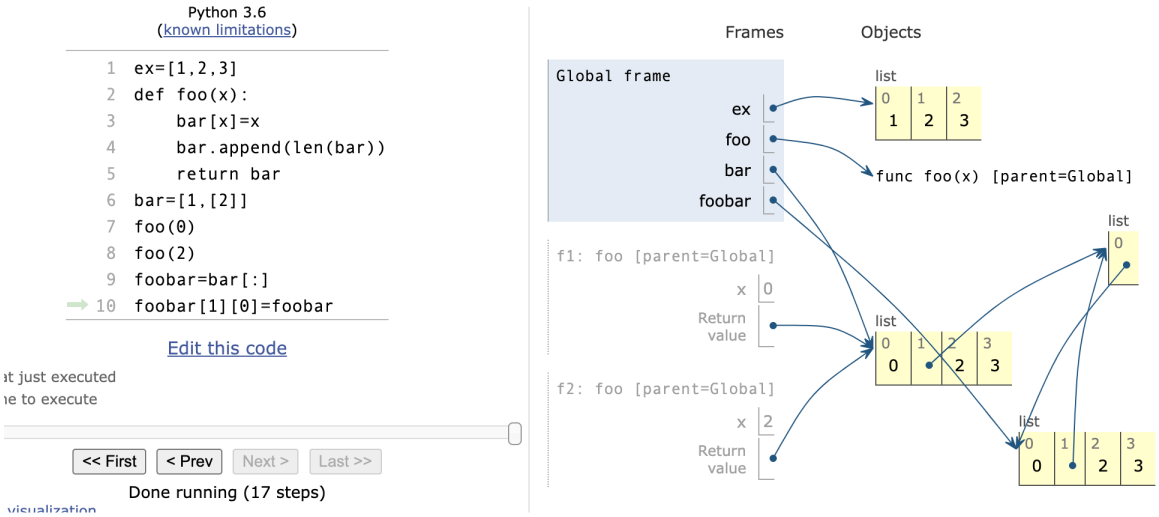
bar=[1,[2]]

foo(0)

foo(2)

foobar=bar[:]

foobar[1][0]=foobar



Some objects in Python, such as lists and dictionaries, are **mutable**, meaning that their contents or state can be changed. Other objects, such as numeric types, tuples, and strings, are **immutable**, meaning they cannot be changed once they are created.内容或状态可以更改——可变，穿件厚不可更改——不变

- `append(el)`: Add `el` to the end of the list. Return `None`.末尾加一个元素，返回 None！el 是一个元素
- `extend(lst)`: Extend the list by concatenating it with `lst`. Return `None`. 末尾延长一个 list，返回 None！lst 要写一个[list]
- `insert(i, el)`: Insert `el` at index `i`. This does not replace any existing elements, but only adds the new element `el`. Return `None`.在 i 处插元素，返回 None！
- `remove(el)`: Remove the first occurrence of `el` in list. Errors if `el` is not in the list. Return `None` otherwise. 拿走第一个对应元素，若没有 Error，返回 None！
- `pop(i)`: Remove and return the element at index `i`.拿走标签 i 上的元素，并返回该元素！

exam practice

eg +=与=+ 的 mutated 与否区别

纯文本

```
>>> x = [1, 2, 3]
>>> y = x
>>> x += [4] #注意这行，x自身加
>>> x
[1,2,3,4] #x自己加
>>> y # y is pointing to the same list as x, which got mutated
[1,2,3,4] #y指向x，也变化

>>> x = [1, 2, 3]
>>> y = x
>>> x = x + [4] # creates NEW list, assigns it to x注意这行新x
>>> x
```

```
[1,2,3,4]
>>> y # y still points to OLD list, which was not mutated仍指向旧x
[1, 2, 3]
```

Lab6 Q1WWPD List

纯文本

```
>>> lst = [5, 6, 7, 8]
>>> lst.append(6)
Nothing   #附加返回None

>>> lst
[5,6,7,8,6]

>>> lst.insert(0, 9)
>>> lst
[9,5,6,7,8,6]

>>> x = lst.pop(2) #注意x=，x即为第二个位置6
>>> lst
[9,5,7,8,6]

>>> lst.remove(x) #remove6
>>> lst
[9,5,7,8]
```

=和 is，[:]新建

纯文本

```
>>> s1 = [1, 2, 3]
>>> s2 = s1 #后续所有对s2的操作也是对s1的操作
>>> s1 is s2
True

>>> s2.extend([5, 6]) #extend（一列）与append区别（一个）
>>> s1[4] #s2指向s1，s1 is s2
6

>>> s1.append([-1, 0, 1]) #append区别（一个）
```

```
>>> s2[5] #s2指向s1，s1 is s2
[-1, 0, 1]

>>> s3 = s2[:] #新建了s3
>>> s3.insert(3, s2.pop(3)) #这里对s2(同时s1)操作了pop
>>> len(s1)
5

>>> s1[4] is s3[6]
True #❓ 因为插入了连到一起了

>>> s3[s2[4][1]] #最后一个[-1, 0, 1]取1即为0，s3[0]=1
1

>>> s1[:3] is s2[:3] #[:] 都新建了，不再是is
False

>>> s1[:3] == s2[:3] 虽然都是新建的，但==,==要求低于is
True
```

Lab6 Q1WWPD is/==

is：两个对象相同?

==：两个对象内容相同?

纯文本

```
>>> a, b = lst, lst[:]
>>> a is lst
True

>>> b == lst
True  #==要求较低

>>> b is lst
False #is要求严苛

>>> lst = [1, 2, 3]
>>> lst.extend([4,5])
>>> lst
[1,2,3,4,5]
```

```
>>> lst.extend([lst.append(9), lst.append(10)])
>>> lst
[1,2,3,4,5,9,10,None,None]  #在lst.append时就已经加9加10，之后extend本身的None和
```

lab10 Q5: Trade

In the integer market, each participant has a list of positive integers to trade. When two participants meet, they trade the smallest non-empty prefix of their list of integers. A prefix is a slice that starts at index 0.

Write a function `trade` that exchanges the first `m` elements of list `first` with the first `n` elements of list `second`, such that the sums of those elements are equal, and the sum is as small as possible. If no such prefix exists, return the string `'No deal!'` and do not change either list. Otherwise change both lists and return `'Deal!'`. A partial implementation is provided.

纯文本

```
def trade(first, second):
    """Exchange the smallest prefixes of first and second that have equal s

    >>> a = [1, 1, 3, 2, 1, 1, 4]
    >>> b = [4, 3, 2, 7]
    >>> trade(a, b) # Trades 1+1+3+2=7 for 4+3=7
    'Deal!'
    >>> a
    [4, 3, 1, 1, 4]
    >>> b
    [1, 1, 3, 2, 2, 7]
    >>> c = [3, 3, 2, 4, 1]
    >>> trade(b, c)
    'No deal!'
    >>> b
    [1, 1, 3, 2, 2, 7]
    >>> c
    [3, 3, 2, 4, 1]
    >>> trade(a, c)
    'Deal!'
    >>> a
    [3, 3, 2, 1, 4]
    >>> b
    [1, 1, 3, 2, 2, 7]
```

```
    >>> c
    [4, 3, 1, 4, 1]
    >>> d = [1, 1]
    >>> e = [2]
    >>> trade(d, e)
    'Deal!'
    >>> d
    [2]
    >>> e
    [1, 1]
    """
    m, n = 1, 1
    equal_prefix = lambda: sum(first[:m])==sum(second[:n])
    while m<=len(first) and n<=len(second) and not equal_prefix(): #要执行la
        if sum(first[:m])<sum(second[:n]):
            m += 1
        else:
            n += 1
    if equal_prefix():
        first[:m], second[:n] = second[:n], first[:m] #交换
        return 'Deal!'
    else:
        return 'No deal!'
```

Lab 10 Q6: Shuffle 洗牌

Define a function `shuffle` that takes a sequence with an even number of elements (cards) and creates a new list that interleaves the elements of the first half with the elements of the second half.输入偶数元素 card，新建 list 前半后半交错

To interleave two sequences `s0` and `s1` is to create a new sequence such that the new sequence contains (in this order) the first element of `s0`, the first element of `s1`, the second element of `s0`, the second element of `s1`, and so on.

> **Note:** If you're running into an issue where the special heart / diamond / spades / clubs symbols are erroring in the doctests, feel free to copy paste the below doctests into your file as these don't use the special characters and should not give an "illegal multibyte sequence" error.

纯文本

```
def shuffle(cards):
    """Return a shuffled list that interleaves the two halves of cards.

    >>> shuffle(range(6))
    [0, 3, 1, 4, 2, 5]
    >>> suits = ['H', 'D', 'S', 'C']
    >>> cards = [card(n) + suit for n in range(1,14) for suit in suits]
    >>> cards[:12]
    ['AH', 'AD', 'AS', 'AC', '2H', '2D', '2S', '2C', '3H', '3D', '3S', '3C'
    >>> cards[26:30]
    ['7S', '7C', '8H', '8D']
    >>> shuffle(cards)[:12]
    ['AH', '7S', 'AD', '7C', 'AS', '8H', 'AC', '8D', '2H', '8S', '2D', '8C'
    >>> shuffle(shuffle(cards))[:12]
    ['AH', '4D', '7S', '10C', 'AD', '4S', '7C', 'JH', 'AS', '4C', '8H', 'JD
    >>> cards[:12]  # Should not be changed
    ['AH', 'AD', 'AS', 'AC', '2H', '2D', '2S', '2C', '3H', '3D', '3S', '3C'
    """
    assert len(cards) % 2 == 0, 'len(cards) must be even'
    half = len(cards) // 2
    shuffled = []
    for i in range(half):
        shuffled.extend([cards[i]]) #extend([])或append()
        shuffled.extend([cards[half+i]])
    return shuffled
```

Q2:Add This Many

Write a function that takes in a value `x`, a value `el`, and a list `s`, and adds `el` to the end of `s` the same number of times that `x` occurs in `s`. **Make sure to modify the original list using list mutation techniques.**用到可变性，s 后加 el 的次数=x 出现的次数

纯文本

```
def add_this_many(x, el, s):
    #法一
    s.extend([el for i in s if i==x])
    #不用写return，可变形直接extend，一次extend前期不影响s
    #[]的写法[表达 for i in list if 筛选条件]

    #法二
    t=0
    for k in s: #用for循环
        if k==x:
            t+=1
    while t>0:
        s.append(el) #注意append后面()里写el即可，不用加[]
        t-=1

    #法三
    for i in range(len(s)):  #for i in range，一锤子买卖len固定，不怕可变性
        if s[i] == x:
             s.append(el) #对自己mutate的小心，尽可能不用for i in s,用i in rang

    #法四
    for a in list(s): #新建list(s)，s一直在变
        if a == x: #第二用a，不能用s[i],不要造成误解
             s.append(el)
```

Lab4 Q2 Insert Items

Write a function which takes in a list `lst`, an argument `entry`, and another argument `elem`. This function will check through each item in `lst` to see if it is equal to `entry`. Upon finding an item equal to `entry`, the function should modify the list by placing `elem` into `lst` right after the item. At the end of the function, the modified list should be returned. 每个 entry，在右侧插入 elem，最后输出 list

Use list mutation to modify the original list. No new lists should be created or returned. 不新建或返回新 list，用 list mutation

纯文本

```
def insert_items(lst, entry, elem):
    #>>> test_lst = [1, 5, 8, 5, 2, 3]
```

```
    #>>> new_lst = insert_items(test_lst, 5, 7)
    #>>> new_lst

    [1, 5, 7, 8, 5, 7, 2, 3]
    i=0
    while i < len(lst): #不能用for i in range长度第一次已设定好，while循环时len(
        if lst[i]==entry:
            lst.insert(i+1,elem) #用insert
            i+=2    #不能用for循环的原因
        else:
            i+=1
    return lst
```

Lab6 Q6: Partial Reverse

Implement this flip operation in code, with the stack of crusts represented as a list starting from the bottom of the stack. Write a function `partial_reverse` to reverse the subset of a list starting from `start` until the end of the list. This reversal should be *in-place*, meaning that the original list is modified 更改原始列. Do not create a new list inside your function, even if you do not return it.

纯文本

```
def partial_reverse(lst, start):
    for i in range(len(lst[start:])//2): #注意//2否则白交换
        lst[i+start],lst[len(lst)-1-i]=lst[len(lst)-1-i],lst[i+start]
        #注意-1，实际最后一个是长度-1，此外必须在一行写完，否则要外加c
```

lab6 Q7: Index Largest

Write a function that takes in a sequence and returns the index of the largest element in the sequence:

纯文本

```
def index_largest(seq):
    #Return the index of the largest element in the sequence.
    法一：
    index=0
    large=seq[0] #追踪最大
```

```
    for i in range(1,len(seq)):
        if seq[i]>large:
            index=i
            large=seq[i]
    return index

    #很6的法二，面向对象用seq.index得到标签，max(seq)也可
    return seq.index(max(seq))
```

lab6 Q8: Pizza Sort

This function takes in a list and sorts its elements in descending order, using only the previous two functions and the built-in `len` function. It is well-known that Leonardo hated iteration, so make sure to use recursion instead. (You may define a helper function if you want.) Lastly, you may use slicing.Keep in mind, however, that slicing returns a new list, but you need to sort the existing list in place.输入列表按照降序排列，用上面的函数(Partial Reverse)和 len，不用迭代用递归，用切片

纯文本

```
def pizza_sort(lst):
    pizza_sort_helper(lst, 0)

def pizza_sort_helper(lst, start):
    if start<len(lst):
        partial_reverse(lst,start+index_largest(lst[start:]))
        partial_reverse(lst, start)
        pizza_sort_helper(lst, start+1)
```