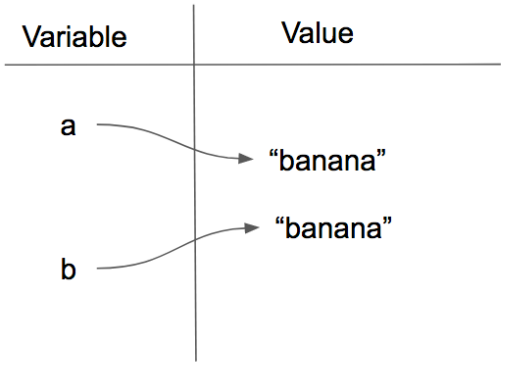
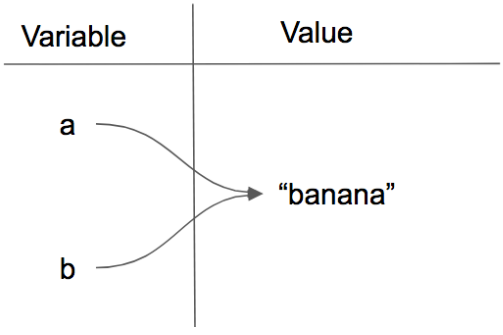
1.4 Sequence Mutation变化

- 两种方法
  - 直接在原来的基础上改 对list可
    - list is mutable , string are not. 所以一个句子里想改个单词是改不了的 就只能创建一个新的再改
      - 例子
        - fruit = ["banana", "apple", "cherry"]
        - print(fruit)
        - fruit[0] = "pear"
        - fruit[-1] = "orange" 也可以换空的东西 alist[1:3] = []这种
        - print(fruit)
        - 输出
        - ['banana', 'apple', 'cherry']
        - ['pear', 'apple', 'orange']
        - 也可以
        - alist = ['a', 'd', 'f']
        - alist[1:1] = ['b', 'c']
        - print(alist)
        - alist[4:4] = ['e']
        - print(alist) 在某个特定位置去放很多东西而不是替换
        - 输出
        - ['a', 'b', 'c', 'd', 'f']
        - ['a', 'b', 'c', 'd', 'e', 'f']
    - 对list里面的element 进行deletion
      - 例子
        - a = ['one', 'two', 'three']
        - del a[1]
        - print(a)
        - alist = ['a', 'b', 'c', 'd', 'e', 'f']
        - del alist[1:5]
        - print(alist)
        - 输出
        - ['one', 'three']

        - ['a', 'f']
- 创建一个新的再改 对string 对tuple
    - 对string
        - 例子
            - greeting = "Hello, world!"
            - newGreeting = 'J' + greeting[1:]
            - print(newGreeting)
            - print(greeting)　　输出就是改过的内容了 只是是不同的variable了
        - 一直延长的案例
            - phrase = "many moons"
            - phrase_expanded = phrase + " and many stars"
            - phrase_larger = phrase_expanded + " litter"
            - phrase_complete = "M" + phrase_larger[1:] + " the night sky."
            - excited_phrase_complete = phrase_complete[:-1] + "!"
    - 对tuple
        - 一旦创建了一个tuple 就无法再更改这个tuple了
- objects and references
    - string 相同的例子
        - a = "banana"　b = "banana" 其实这两个refer去的string长得都一样 但不知道是不是同一个string 这两种情况不确定是哪种
        -





    - 所以test
    - a = "banana"
    - b = "banana"
    - print(a is b) 如果a和b指向同一个string 那会pring True
    - 输出True
    - 其实python做的真正的筛查是
    - a = "banana"
    - b = "banana"
    - print(id(a))
    - print(id(b)) 每个变量都会分到一个id 就有点像身份证号码，指示你到底是unique的还是跟别人一样
    - 输出 2 2 意味着这两个是同一个id if id(a) == id(b). 这里就是a和b互为 aliases of each other 别名 化名 就跟fwq叫林唯这两个都指的是同一个人 但是换了个名字
- list不同的例子
    - a = [81,82,83]
    - b = [81,82,83]
    - print(a is b)
    - print(a == b)
    - print(id(a))
    - print(id(b))
    - 输出
    - False
    - True
    - 3
    - 4 a和b两个list有着相同的value 但是是不同的object 所以a is b的时候显示False 但是又因为内容一样 所以a==b是True 但本质python给了这两个list 不同的id
    - 如果想让a和b变成一样的东西 只需要再写一步 a=b让他们互为对方的alias
        - a = [81,82,83]
        - b = [81,82,83]
        - print(a is b)
        - b = a
        - print(a == b)
        - print(a is b)
        - b[0] = 5
        - print(a) 这里虽然我没有直接改变a 因为改变了b所以最后输出a的时候也是会改变

        - 输出 False True True [5, 82, 83]
    - 但是如果只是想复制一个list而不是alias 就叫clone 用slice
        - a = [81,82,83]
        - b = a[:] # make a clone using slice
        - print(a == b)
        - print(a is b)
        - b[0] = 5
        - print(a)
        - print(b)
        - 输出
        - True
        - False
        - [81, 82, 83]
        - [5, 82, 83]
- Methonds on lists
    - append 附加 贴上 盖章 在最后一个的位置
        - append不会改变a属于某list的这个箭头 只会改变list中的值
        - mylist = []
        - mylist.append(5)
        - mylist.append(27)
        - mylist.append(3)
        - mylist.append(12)
        - print(mylist) 一步一步从空的list里加5，27，3，12加在后面
        - 输出 [5,12,27,3]
    - append和concatenate 连接
        - concatenate： newlist = origlist + ["cat"]
        - 一个tips是 对list不要用+=这个符号 就老老实实+ 因为如果有alias的时候很容易把所有的都替换了 到时候自己搞不清楚
        - concatenate的例子
            - st = "Warmth"
            - a = []
            - b = a + [st[0]]
            - c = b + [st[1]]
            - d = c + [st[2]]
            - e = d + [st[3]]

        - f = e + [st[4]]
        - g = f + [st[5]]
        - print(g)
        - 输出['W', 'a', 'r', 'm', 't', 'h']
        - 如果append的话就是
        - st = "Warmth"
        - a = []
        - a.append(st[0])
        - a.append(st[1])
        - a.append(st[2])
        - a.append(st[3])
        - a.append(st[4])
        - a.append(st[5])
        - print(a)但是从头到尾只有a这个list 而不像上面有abcdefg
        - 每个加五进list的例子
            - alist = [4,2,8,6,5] blist = [ ] for item in alist: blist.append(item+5) print(blist)
- insert 你告诉他加哪里就加哪里 不像append一样只能加最后,
    - mylist.insert(1, 12) 在1的位置插入12
    - print(mylist)
- count数数
    - print(mylist.count(12))在mylist这个list里数12有几个
- index 索引
    - print(mylist.index(3)) 找到3的位置去print out那个值
- reverse
    - mylist.reverse() 前后颠倒
- sort
    - 把元素组合按照a-z的顺序
    - mylist.sort()
- 颠倒和sort结合的例子
    - winners = ['Alice Munro', 'Alvin E. Roth', 'Kazuo Ishiguro', 'Malala Yousafzai', 'Rainer Weiss', 'Youyou Tu']
    - z_winners=sorted(winners,reverse=True)
- remove跟之前del的区别 remove看的是value del看的是位置
    - mylist.remove(5) 删掉5这个元素

- pop 会让list change
  - lastitem = mylist.pop()把最后一个元素拎出来另外组一个list命名为lastitem
  - pop同时也可以限定位置
  - print(lastitem)
  - print(mylist)
- mutator means that the list is changed by the method but nothing is returned；
  hybrid method is one that not only changes the list but also returns a value as its result；
  Finally, if the result is simply a return, then the list is unchanged by the method.

Method	Parameters	Result	Description
append	item	mutator	Adds a new item to the end of a list
insert	position, item	mutator	Inserts a new item at the position given
pop	none	hybrid	Removes and returns the last item
pop	position	hybrid	Removes and returns the item at position
sort	none	mutator	Modifies a list to be sorted
reverse	none	mutator	Modifies a list to be in reverse order
index	item	return idx	Returns the position of first occurrence of item
count	item	return ct	Returns the number of occurrences of item
remove	item	mutator	Removes the first occurrence of item

- append sort reverse 这几个都return NONE 已经change了list且不产生新的list 所以会失去整个list content...?

- Methods on string
  - upper和lower 全大写和全小写
    - 例子
      - ss = "Hello, World"
      - print(ss.upper())
      - tt = ss.lower()
      - print(tt)
      - print(ss)
      - 输出
      - HELLO, WORLD
      - hello, world
      - Hello, World
  - strip 把string前后的空格都去掉 但是保留中间的空格
    - ss = " Hello, World "

    - els = ss.count("l")
    - print(els)
    - print("***"+ss.strip()+"***")
    - news = ss.replace("o", "***")
    - print(news)
    - 输出
    - 3
    - ***Hello, World***
    - Hell***, W***rld
- replace 用某个代替某个
    - 看上面例子的
    - news = ss.replace("o", "***")
- string format method
    - 例子
        - name = "Rodney Dangerfield"
        - score = -1 # No respect!
        - print("Hello " + name + ". Your score is " + str(score))
        - 这样其实很麻烦 而且包含 hard coding
        - 改进
        - scores = [("Rodney Dangerfield", -1), ("Marlon Brando", 1), ("You", 100)] 弄一个list 包含很多tuple
        - for person in scores:
        - name = person[0]
        - score = person[1]
        - print("Hello {}. Your score is {}.".format(name, score)) curly括号就是要填充的东西 然后.format
        - 比如说要写个 hello___！ 然后需要你输入自己的名字 自己会跳出来hello谁谁谁
        - person = input('Your name: ')
        - greeting = 'Hello {}!'.format(person) ‘’里面的内容是到时候希望讲的话 曲括号{}中是到时候会要放进来的东西.format(中间填variable name)
        - print(greeting)
    - 还有个例子是算价钱的
        - origPrice = float(input('Enter the original price: $'))
        - discount = float(input('Enter discount percentage: '))
        - newPrice = (1 - discount/100)*origPrice

        - calculation = '${} discounted by {}% is ${}.'.format(origPrice, discount, newPrice) 括号里的东西就是 曲括号里要填的东西 分别是三个变量
        - print(calculation)
        - 如果想要输出的是几个小数的话就改一下其中的一点东西calculation = '${:.2f} discounted by {}% is ${:.3f}.'.format(origPrice, discount, newPrice)
        - :.2f即为两位小数 如果要三位的话就:.3f
    - 用format的好处
        - name = "Sally"
        - greeting = "Nice to meet you"
        - s = "Hello, {}. {}."
        - print(s.format(name,greeting)) # will print Hello, Sally. Nice to meet you.
        - print(s.format(greeting,name)) # will print Hello, Nice to meet you. Sally.
        - print(s.format(name)) # 2 {}s, only one interpolation item! Not ideal.
    - 这里有个说明不是很看得懂
        - 就是如果希望最后呈现的是曲括号{5,9}这样代表数字的集合 但是其实在format用法中{}有其特殊的含义，所以用 \{\{ and \}\} 来代替
        - a = 5b = 9setStr = 'The set is \{\{\{}, {}}}.'.format(a, b)print(setStr)
    -

Method	Parameters	Description
upper	none	Returns a string in all uppercase
lower	none	Returns a string in all lowercase
count	item	Returns the number of occurrences of item
index	item	Returns the leftmost index where the substring item is found and causes a runtime error if item is not found
strip	none	Returns a string with the leading and trailing whitespace removed
replace	old, new	Replaces all occurrences of old substring with new
format	substitutions	Involved! See String Format Method, below

- accumulator pattern with lists
    - 例子
        - nums = [3, 5, 8]
        - accum = []
        - for w in nums:
        - x = w**2
        - accum.append(x)

    - print(accum)
    - 输出 [9,25,64]
    - 如果要加的话就要写成accum = accum + [x]了
    - 相似
    - verbs = ["kayak", "cry", "walk", "eat", "drink", "fly"]
    - ing=[]
    - for word in verbs:
    - x=word+"ing"
    - ing.append(x)
  - list中每个数字都加五但是不创建新的list
    - numbs = [5, 10, 15, 20, 25]
    - numbs=list(map(lambda x:x+5,numbs))
    - 网上看来的 并不知道什么意思
- accumulator pattern with strings
  - 例子
    - s = input("Enter some text")
    - ac = ""
    - for c in s:
    - ac = ac + c + "-" + c + "-"
    - print(ac)
    - 随便输入一个单词 创造一个空的string 命名为ac 对每个字母in s 让其变身双倍 具体看program
    - 比如我输入dfg 输出的是d-d-f-f-g-g-
  - 易错
    - s = "ball"
    - r = ""
    - for item in s:
    - r = item.upper() + r
    - print(r)
    - 输出的其实是LLAB the order is reversed due to the order of the concatenation不知道为什么
  - 有一个其实还没想通
    - str1 = "I love python" # HINT: what's the accumulator? That should go here.
    - chars = []
    - for character in str1:

        - chars.append(character)
    - 放每个的首字母
        - colors = ["Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Violet"]
        - initials = []
        - for color in colors:
        - initials.append(color[0])
        - print(initials)
    - 删去首字母是PBT的单词
        - colors = ["Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Violet", "Purple", "Pink", "Brown", "Teal", "Turquois", "Peach", "Beige"]
        - for position in range(len(colors)):
        - color = colors[position] 这里好像会有问题
        - print(color)
        - if color[0] in ["P", "B", "T"]:
        - del colors[position]
        - print(colors)
- strategy去选择什么字母

Phrase	Accumulation Pattern
how many	count accumulation
how frequently	
total	sum accumulation
a list of	list accumulation
concatenate	string accumulation
join together	

-