Concatenation and Repetition of sequence types

Expressions for sequential types

```
seq1 + seq2   concatenation of two sequence types
seq1 * n      repetition of the sequence n times
```

```
s1="music"
s2="jazz"
s1+s2       #string concatenation 'musicjazz'
s1+' '+s2   #'music jazz'
s1*3        #string repetition 'musicmusicmusic'
```

```
L1=[1,2,3,4]
L2=[5,6,7,8]
L1+L2          #list concatenation [1, 2, 3, 4, 5, 6, 7, 8]
L1*2           #list repetition [1, 2, 3, 4, 1, 2, 3, 4]
```

```
s1+L # cannot concatenate two different sequential types
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "list") to str
```

Functions of sequence type

len(seq) returns the length (number of elements) of a sequence

min(seq) returns smallest item of seq - works for homogeneous items

max(seq) returns largest item of seq - works for homogeneous items

sum(seq) returns the sum – works for list or tuple of numbers

```
L=[1,2,5,4,100]
T=(3,7.8,10000)
max(L) # 100
min(T) # 3
sum(L) # 112

s='abghtyu'
max(s) # alphabetically 'y'
len(s) # 7
len(L) # 5
```

Methods of sequence types

There are some functions and methods (methods are functions applied by using a dot) that work on a sequence type: string, list, and tuple

`seq.count(elem)` returns the number of occurrences of elem in the sequence

`seq.index(elem)` returns the index of the first occurrence of elem

```
s1="this is a string"
s1.count("i") # 3
s1.index('h') # 1
```

```
L=["cat",'cat','dog','elephant']
L.count("cat") # 2
L.index("cat") # 0
```

Common operations for sequence data types: strings, lists and tuples

seq and seq1 can be a list, a string or a tuple

Operation	Result
seq + seq1	Concatenation of seq and seq1
seq*n	Repetition of seq n times
seq[i]	Return item at index i
seq[i:j]	Slice of seq from i to j-1
seq[i:j:k]	Slice of seq from i to j-1 with step k
len(seq)	Total number of elements in seq
seq.count(x)	Number of occurrences of x in seq
seq.index(x)	Return index of x in seq
min(seq)	Return smallest item of seq
max(seq)	Return largest item of seq
sum(seq)	Return the sum – works on list or tuple of numbers

Mutability

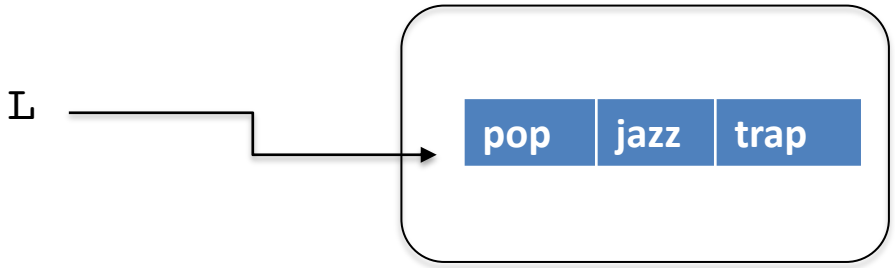
In Python, every object (data type) has these three attributes:

- **Identity:** the address that the object refers to in the computer's memory. *id()* function
- **Type:** the kind of object that is created. integer, list, string etc. *type()* function
- **Value** stored by the object. For example – List=[1,2,3] would hold the numbers 1,2 and 3.

Objects whose values can change are said to be **mutable**

Objects whose values are unchangeable once they are created are called **immutable**.

Lists are mutable Objects



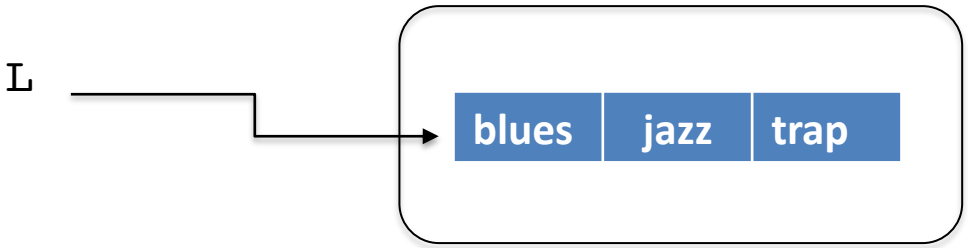

```
L=['pop','jazz','trap']
print(id(L))
print(type(L))
print(L) #values are the elements: 'pop' 'jazz' 'trap'
```

List - Mutability and Methods

Objects whose value can change are said to be **mutable**

```
object[index]=new_value  #item assignment
```

```
L=['pop','jazz','trap']
id(L) #140265716380096
L[0]='blues'
id(L) #140265716380096
```




```
list.append(item) #append method adds an item to the end
```

```
L.append('classic') #returns None
id(L)) #140265716380096
```

List Methods

These list methods return the default None.

```
list.reverse()        Reverse the elements of the list in place.

list.append(item)     Add an item to the end of the list.

list.insert(i,item)   Insert item at index i.

list.remove(item)     Remove the first occurrence of item

list.extend(list1)    Extend the list by joining list1
```

but let's see the pop method

```
list.pop(i)       Remove the item at index i and return that item.
list.pop()        Remove and return the last item in the list.
```

Get info on list methods
https://docs.python.org/3.1/tutorial/datastructures.html

List Method examples


```
Lzoo = ["pangolin", "chicken", "lion"]

Lzoo.append("elephant")
print(Lzoo)
['pangolin', 'chicken', 'lion', 'elephant']

Lzoo.insert(1,"cobra")
print(Lzoo)
['pangolin', 'cobra', 'chicken', 'lion', 'elephant']

Lzoo.extend(["dog", "cat"])
print(Lzoo)
['pangolin', 'cobra', 'chicken', 'lion', 'elephant', 'dog', 'cat']


Lzoo.remove("pangolin")
print(Lzoo)
['cobra', 'chicken', 'lion', 'elephant', 'dog', 'cat']

x=Lzoo.pop(1)
print(Lzoo)
print(x)
['cobra', 'lion', 'elephant', 'dog', 'cat']
chicken
```

List - Delete items with del command

The del statement can also be used to remove an item or a range of items

```
del list[index]
del list[start:end]
del list[start:end:step]
```

```
L=['blues', 'soul', 'trap', 'classic', 'rock', 'punk' ]

del L[0]   #delete element at index 0
print(L)
['soul', 'trap', 'classic', 'rock', 'punk' ]


del L[2:]  #delete a range of elements by using slicing
print(L)
['soul', 'trap']
```

Dictionary methods

Just like Lists, **Dictionaries are** mutable . This means they can be changed after they are created.

`dict.pop(key)`	Remove the key-value and return that value
`dict.popitem()`	Remove the last key-value pair and return it as tuple
`dict.update(dict2)`	Update dictionary dict by adding key:value of dict2

```
D={"apple":2,"cherry":4, "fig":10, "banana":3}

t=D.popitem()
t #('banana', 3)
D # {'apple': 2, 'cherry': 4, 'fig': 10}

x=D.pop("apple")
D #{'cherry': 4, 'fig': 10}
x # 2

D.update({"pear":6, "orange":3})
D #{'cherry': 4, 'fig': 10, 'pear': 6, 'orange': 3}
```

Dictionary– add and remove key:value

```
d[key]=value   #add a new key/value pairs
               #assign a new value to an existing key

del d[key1]   #remove the key1 and the associated value
```

```
Dzoo= {"pangolin" : 5, "sloth" : 3, "tiger" : 20}
```

```
Dzoo["snake"]=6  #add key "snake" and associated value 6
{'pangolin': 5, 'sloth': 3, 'tiger': 20, 'snake': 6}

Dzoo["sloth"]=4 #assign a new value to key "sloth"
{'pangolin': 5, 'sloth': 4, 'tiger': 20, 'snake': 6}
```

```
del Dzoo["tiger"] #remove the key:value pair "tiger":20
{'pangolin': 5, 'sloth': 4, 'snake': 6}
```

More Dictionary methods

```
dict.clear()        Remove all the elements from the dictionary
dict.copy()         Return a copy of the dictionary
dict.get(key)       Return the value of the specified key
```

The length len() of a dictionary return the number of key-value pairs

`len(Dzoo)`

Get info on dictionary methods

https://realpython.com/python-dicts/

String type - Immutability and Methods

Objects whose values are unchangeable once they are created are called **immutable**.

```
s='Hello'
```

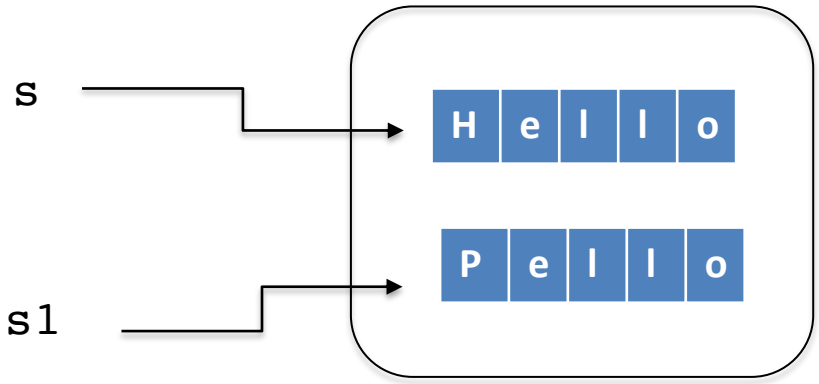
Let's try to change H with P and try to use item assignment.

```
s[0]='P'
TypeError: 'str' object does not support item assignment
```

If you apply a string method, a new string object is created, which you can assign to a new variable.

Let's try to apply the string method replace

```
s1=s.replace('H', 'P')
print(id(s1)
print(id(s))
```



Some string methods

String method examples:

```
s="I like school"


s1=s.upper()  #upper method takes no arguments
print(s1)


s2=s.replace("school","play") #replace takes two arguments
print(s2)
```

More string methods can be found here
https://docs.python.org/3/library/stdtypes.html#string-methods

Combining String Methods and indexing, slicing

You can apply more than one string method on the same string in one command line. Next method applies to the output of the previous method.

```
s1=s.upper().replace("SCHOOL","PLAY")
print(s1)
```

You can slice

```
print(s.upper()[7:])

print(s.upper().replace("SCHOOL","PLAY")[7:])

s2=s.upper().replace("SCHOOL","PLAY")[7:]
print(s2)
```

Tuple type - Immutability

A tuple type support the index and count methods! A tuple is an immutable type

```
T=(42,[1,2,3],'hello')
T[1]="Hello"
TypeError: 'tuple' object does not support item assignment
```

If an element of a tuple is mutable, that element can change. However, the tuple is considered immutable, because the collection of objects it contains cannot be changed.
So, immutability is not strictly the same as having an unchangeable value, it is more subtle.

```
T=(42,[1,2,3],'hello')
id(T) # 140645552536832

T[1].append('pop')
id(T) # 140645552536832

print(T)
(42, [1, 2, 3, 'pop'], 'hello')
```