Lecture 6, 25 August 2022

Mutable and immutable values

`l.append(x)` vs `l = l + [x]`

- `l.append(x)` modifies `l` in place, returns `None`
- `l = l + [x]` creates a new list `l`

In [1]:
```
l1 = [1,2,3]
l2 = l1
l1.append(4)
```

In [2]:
```
(l1,l2)
```

Out[2]:
```
([1, 2, 3, 4], [1, 2, 3, 4])
```

In [3]:
```
l3 = [1,2,3]
l4 = l3
l3 = l3+[4]
```

In [4]:
```
(l3,l4)
```

Out[4]:
```
([1, 2, 3, 4], [1, 2, 3])
```

In [5]:
```
l4[0]=5
```

In [6]:
```
l3,l4
```

Out[6]:
```
([1, 2, 3, 4], [5, 2, 3])
```

In [7]:
```
l1 = [1,2,3]
l2 = l1     # l2 is 'aliased' to l1
l1[0] = 4   # Updates value for both l1 and l2
l1,l2
```

Out[7]:
```
([4, 2, 3], [4, 2, 3])
```

In [8]:
```
l1 = [1,2,3]
l2 = l1
l3 = l1     # l1, l2, l3 are all the same list
l2[0] = 4   # Doesn't matter which copy you update
l1,l2,l3    # All names report the updated value
```

Out[8]:
```
([4, 2, 3], [4, 2, 3], [4, 2, 3])
```

In [9]:
```
l1 = [1,2,3]
l2 = l1
l1[0] = 4    # Changes l2[0] as well
l1 = l1 + [] # Now l1 is a separate list from l2, same as l1 = l1[:]
l1[0] = 1    # Does not update l2
l3 = l2[:]   # Full slice, faithful copy
l3[0]=7      # Does not affect l2
l1,l2,l3
```

Out[9]:
```
([1, 2, 3], [4, 2, 3], [7, 2, 3])
```

Depending on the need, it may be useful to modify a list in place or return a modified list without changing the original

Sorting

- `l.sort()` sorts a list in place
- `sorted(l)` returns a sorted copy of the list, leaving the original unchanged

In [10]:
```
mylist = [7,3,1,5,6]
```

In [11]:
```
mylist.sort()  # sorts in place
```

In [12]:
```
mylist
```

Out[12]:
```
[1, 3, 5, 6, 7]
```

In [13]:
```
mylist = [7,3,1,5,6]
sorted(mylist)  # Does not modify argument, returns sorted list
```

Out[13]:
```
[1, 3, 5, 6, 7]
```

In [14]:
```
mylist
```

Out[14]:
```
[7, 3, 1, 5, 6]
```

In Python (but not in other languages), *methods* like `mylist.sort()` can also be called as functions, in this case `list.sort(mylist)`. Need to say `list.sort()` to tell Python where to find the function `sort()`.

In [15]:
```
mylist = [7,3,1,5,6]
```

In [16]:
```
list.sort(mylist)
```

In [17]:
```
mylist
```

Out[17]:
```
[1, 3, 5, 6, 7]
```

Equality

- `x == y` checks if `x` and `y` have the same value --- but need not be the same "box" in memory for mutable values
- `x is y` checks if `x` and `y` point to the same "box" in memory
- if `x is y` is `True`, then necessarily `x == y` is `True`, but not vice versa

In [18]:
```
l1 = [3,4,5]
l2 = l1[:]
l3 = l1
l1 == l2, l1 == l3, l1 is l2, l1 is l3
```

Out[18]:
```
(True, True, False, True)
```

In [19]:
```
x = 7
y = x
# x = 8
x is y, x == y
```

Out[19]:
```
(True, True)
```

In [20]:
```
x = 8
y = x
x = 7
x = 8
x is y
```

Out[20]:
```
True
```

Mutable and immutable values

- More accurately, these are properties of values, not names/variable
- Mutability - can the value being pointed to change in place?
- There is only "one copy" of every immutable value
- If we update `x` from 3 to 4, `x` points to the one value of 4 instead of the one value of 3

In [21]:
```
x = 3
y = x
x is y, x == y
```

Out[21]:
```
(True, True)
```

In [22]:
```
x = 3
y = x
x = 4
x is y, x == y
```

Out[22]:
```
(False, False)
```

In [23]:
```
x = 3
x is y, x == y
```

Out[23]:
```
(True, True)
```

- Even in a list, you have to interpret mutability correctly
- If `l[i]` points to an immutable value and you make another name `x` point to the same value, updating `l[i]` will not update `x`
- Only when one list is aliased to another will updating `l[i]` in the first list also reflect in the same position in the second list

In [24]:
```
l1 = [7,8]
l2 = l1
x = l1[0]  # l1[0] is now like an individual variable y
l1[0] = 9
x is l1[0], x == l1[0], l1[0] is l2[0]
```

Out[24]:
```
(False, False, True)
```

Functions and parameters

- Pass a mutable value, then it can updated in the function
- Immutable values will be copied

In [25]:
```
def factorial(n):
  answer = 1
  while (n > 0):
    answer = answer * n
    n = n - 1
  return(answer)
  # Loop computes n * (n-1) * .... * 1
```

In [26]:
```
m = 8
z = factorial(m)  # factorial(n) "begins with" n = m
m, z              # Argument m is unaffected by n being modified in function
```

Out[26]:
```
(8, 40320)
```

Mutability and functions

It is useful to be able to update a list inside a function --- e.g. sorting it

- Built in list functions update in place
- `l.append(v)` -> in place version of `l = l+[v]`
- `l.extend(l1)` -> in place version of `l = l + l1`

In [27]:
```
l1 = [0]
l2 = l1
l1.append(1)
l1.extend([4,3,6,5])
l3 = sorted(l1)
l1, l2, l3
```

Out[27]:
```
([0, 1, 4, 3, 6, 5], [0, 1, 4, 3, 6, 5], [0, 1, 3, 4, 5, 6])
```

Strings

- Text
- Sequence of characters, operations are similar to a list
- But immutable
- Denote a string using single, double or triple quotes

In [28]:
```
x = "hello"
y = 'hello'
b = "Madhavan's book"
statement = '"Hello", he said'
mixedstr = '''"Hello", he said, "where is Madhavan's book?"'''
x, y, b, mixedstr
```

Out[28]:
```
('hello',
 'hello',
 "Madhavan's book",
 '"Hello", he said, "where is Madhavan\'s book?"')
```

- Use positions, slices, concatenation etc as for lists
- No separate single character type; a single character is a string of length 1

In [29]:
```
y[4][0][0], x[2:4], x+' '+b, x[-10:10], y[0]+y[1:]
```

Out[29]:
```
('o', 'll', "hello Madhavan's book", 'hello', 'hello')
```

Slice update in a list

- Can update a slice in a list
  - `l[i] = v`
  - `l[i:j] = l2`
  - This can grow or shrink the list

In [30]:
```
l = list(range(10))
l[2:5] = [11]
l
```

Out[30]:
```
[0, 1, 11, 5, 6, 7, 8, 9]
```

In [31]:
```
l = list(range(10))
l[4:7] = [11,12,13,14,15]
l
```

Out[31]:
```
[0, 1, 2, 3, 11, 12, 13, 14, 15, 7, 8, 9]
```

- Strings are immutable
  - Change `hello` to `helps`

In [32]:
```
h = 'hello'
h[3:5] = 'ps'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_131889/1400047818.py in <module>
      1 h = 'hello'
----> 2 h[3:5] = 'ps'

TypeError: 'str' object does not support item assignment
```

- Use slices, concatenation to reconstruct a new string and reassign to the name

In [33]:
```
h = h[0:3] + 'ps'
h
```

Out[33]:
```
'helps'
```

Basic input and output

- Take input from the keyboard
- Print output to the screen

In [34]:
```
x = input("Please enter a number")
```

```
Please enter a number44
```

In [35]:
```
x
```

Out[35]:
```
'44'
```

In [36]:
```
x + 52
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_131889/88780871.py in <module>
----> 1 x + 52

TypeError: can only concatenate str (not "int") to str
```

Type conversion

- int(s) converts a string s to an int, if it is possible

In [37]:
```
int(x) + 52
```

Out[37]:
```
96
```

- Any type name can be used as a type converter

In [38]:
```
list("hello")
```

Out[38]:
```
['h', 'e', 'l', 'l', 'o']
```

- Type conversion works only if argument is of a sensible type
- For instance, `int(x)` requires `x` to be a valid integer

In [39]:
```
int("5.2")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_131889/702551320.py in <module>
----> 1 int("5.2")

ValueError: invalid literal for int() with base 10: '5.2'
```

- Optional second argument indicates the base for `int` conversion
- For instance, in base 16, `a` to `f` are legal

In [41]:
```
int("a7",16)
```

Out[41]:
```
167
```

Output

- `print(x1,x2,...,xn)`
- Implicitly each `xi` is converted to `str(xi)`
- Use `sep=` to modify default space separating values
- Use `end=` to modify default new line after each print

In [42]:
```
x = 'a'
print(7,x)
print(8,x)
```

```
7 a
8 a
```

In [43]:
```
x = 'a'
print(7,x,sep=",",end="::")
print(8,x)
```

```
7,a::8 a
```

Tuples

- `(x1,x2,x3)` - round brackets, not square
- Immutable sequence (unlike a list)
- Otherwise manipulate using indices, slices etc

In [44]:
```
x = (1,2,3,4)
x[0], x[2:], x[2:][0]
```

Out[44]:
```
(1, (3, 4), 3)
```

In [45]:
```
x[0] = 7
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_131889/2771985514.py in <module>
----> 1 x[0] = 7

TypeError: 'tuple' object does not support item assignment
```

Multiple assignment using tuples

In [46]:
```
x,y = 3,5  # same as (x,y) = (3,5)
```

In [47]:
```
y, x
```

Out[47]:
```
(5, 3)
```

Useful to initialize many things at the start of a function

In [48]:
```
i,j,k,factorlist = 0,0,0,[]
i,j,k,factorlist
```

Out[48]:
```
(0, 0, 0, [])
```

Exchange the values of two variables

- Can we write a function `swap(a,b)` that exchanges the values of its (immutable) arguments?
- Can only do something like `a,b = swap(a,b)`

- To swap `x` and `y`, normally we need an intermediate temporary value `tmp`

```
tmp = y
y = x
x = tmp
```

- In Python, tuple assignment works!

```
(x,y) = (y,x)
```

Dictionaries

- A list is a collection indexed by position
- A list can be thought of as a function $f : \{0, 1, \ldots, n-1\} \rightarrow \{v_0, v_1, \ldots, v_{n-1}\}$
  - A list maps positions to values
- Generalize this to a function $f : \{x_0, x_1, \ldots, x_{n-1}\} \rightarrow \{v_0, v_1, \ldots, v_{n-1}\}$
  - Instead of positions, index by an abstract *key*
- **dictionary**: maps keys, rather than positions, to values
- Notation:
  - `d = {k1:v1, k2:v2}`, enumerate a dictionary explicitly
  - `d[k1]`, value in dictionary `d1` corresponding to key `k1`
  - `{}`, empty dictionary (`[]` for lists, `()` for tuples)
  - Keys must be immutable values

In [49]:
```
names = ["Abha","Bunty"]
#bdays = {}
bdays = {"Abha":"03-05-2001", "Bunty":"17-10-1999"}
```

In [50]:
```
bdays["Bunty"]
```

Out[50]:
```
'17-10-1999'
```

- Accessing a non-existent key results in `KeyError`
- Analogous to `IndexError` for invalid position in a list

In [51]:
```
bdays["Chitra"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/tmp/ipykernel_131889/3028590629.py in <module>
----> 1 bdays["Chitra"]

KeyError: 'Chitra'
```

In [52]:
```
l = [0,1,2]
l[3] = 3
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_131889/628173498.py in <module>
      1 l = [0,1,2]
----> 2 l[3] = 3

IndexError: list assignment index out of range
```

- Assigning to non-existent key creates a new key-value pair
  - Unlike lists, where we cannot create a new position by assigning outside the current list
- `d[k] = v` creates `k` if it does not exist, updates the value at `d[k]` if it does exist

In [57]:
```
bdays["Chitra"] = "13-08-2000"
bdays
```

Out[57]:
```
{'Abha': '03-05-2001',
 'Bunty': '17-10-1999',
 'Newperson': '12-10-2000',
 0: [77],
 'Chitra': '13-08-2000'}
```

In [58]:
```
bdays["Chitra"] = "13-09-2000"
bdays
```

Out[58]:
```
{'Abha': '03-05-2001',
 'Bunty': '17-10-1999',
 'Newperson': '12-10-2000',
 0: [77],
 'Chitra': '13-09-2000'}
```

In [59]:
```
newbdays = bdays
newbdays["Newperson"] = "12-10-2000"
```

In [60]:
```
bdays, newbdays
```

Out[60]:
```
({'Abha': '03-05-2001',
  'Bunty': '17-10-1999',
  'Newperson': '12-10-2000',
  0: [77],
  'Chitra': '13-09-2000'},
 {'Abha': '03-05-2001',
  'Bunty': '17-10-1999',
  'Newperson': '12-10-2000',
  0: [77],
  'Chitra': '13-09-2000'})
```

- Any immutable value can be a key
- No requirement that all keys (or values) have a uniform type

In [61]:
```
bdays[0] = [77]
bdays
```

Out[61]:
```
{'Abha': '03-05-2001',
 'Bunty': '17-10-1999',
 'Newperson': '12-10-2000',
 0: [77],
 'Chitra': '13-09-2000'}
```

- Key must be an immutable value
  - List cannot be used as a key

In [62]:
```
bdays[[1]] = 77
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_131889/890687865.py in <module>
----> 1 bdays[[1]] = 77

TypeError: unhashable type: 'list'
```

- Value at a key can be a list
- Use multiple subscripting to access inner components
- Similar to nested lists

In [63]:
```
bdays[1] = [77,88]
bdays, bdays[1][1]
```

Out[63]:
```
({'Abha': '03-05-2001',
  'Bunty': '17-10-1999',
  'Newperson': '12-10-2000',
  0: [77],
  'Chitra': '13-09-2000',
  1: [77, 88]},
 88)
```

- Likewise, value can be a nested dictionary

In [64]:
```
brothers = {}
brothers["ahmed"] = {"first":"abdul", "second":"salman"}
brothers, brothers["ahmed"]["second"]
```

Out[64]:
```
({'ahmed': {'first': 'abdul', 'second': 'salman'}}, 'salman')
```

Operating on dictionaries

- How do we run through all entries in a dictionary - the equivalent of `for x in l`?
- `d.keys()`, `d.values()` generate sequences corresponding to the keys and values of `d`, respectively
- Like `range()` these are not directly lists, use `list(d.keys())` if you want a list

In [65]:
```
bdays.keys()  # Not quite a list, a bit like range()
```

Out[65]:
```
dict_keys(['Abha', 'Bunty', 'Newperson', 0, 'Chitra', 1])
```

In [66]:
```
for name in bdays.keys():
  print(name)
```

```
Abha
Bunty
Newperson
0
Chitra
1
```

- Shortcut, can omit `.keys()` when iterating

In [67]:
```
for name in bdays:
  print(name)
```

```
Abha
Bunty
Newperson
0
Chitra
1
```

In [68]:
```
namelist = list(bdays.keys())
namelist
```

Out[68]:
```
['Abha', 'Bunty', 'Newperson', 0, 'Chitra', 1]
```

In [69]:
```
list(bdays.values())  # The values in a dictionary
```

Out[69]:
```
['03-05-2001', '17-10-1999', '12-10-2000', [77], '13-09-2000', [77, 88]]
```

- In what order does `d.keys()` list the keys?
- In theory, this order is arbitrary and you should not make any assumptions
- In practice, from some recent version of Python (3.6?) keys are listed in the order added
- If dictionary keys are of the same type, use `sorted(d.keys())` to get them in sorted

In [70]:
```
d = {}
d['a'] = 7
d['c'] = 9
d['b'] = 8
```

In [71]:
```
list(d.keys()), sorted(d.keys())
```

Out[71]:
```
(['a', 'c', 'b'], ['a', 'b', 'c'])
```

In [72]:
```
for name in sorted(d.keys()):
  print(d[name])
```

```
7
8
9
```