Lecture 6, 07 October 2021

Difference between `l.append(x)` and `l = l + [x]`

- `l.append(x)` modifies `l` in place
- `l = l + [x]` creates a new list `l`

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

```
([4, 2, 3], [4, 2, 3])
```

```
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
            # All copies report the updated value
```

Saved successfully!

```
, 3])
```

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

```
([1, 2, 3], [4, 2, 3], [7, 2, 3])
```

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

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

```
(True, True, False, True)
```

```
x = 7
y = x
x = 8
x is y, x == y
```

```
(False, False)
```

```
zerolist = [0,0]
lz = [zerolist,zerolist]
lz[0][0] = 1
lz, zerolist
```

```
([[1, 0], [1, 0]], [1, 0])
```

Passing parameters to functions

- `def f(a,b,c): ...` When `f` is called as `f(x,y,z)`, it as though we start with assignments `a = x`, `b = y`, `c = z`

```
def f(a,b,c):
  a = a + a
  b = b + b
  c = c + c
  return(a+b+c)

x = 10
y = 20
z = 30
f(x,y,z),x,y,z
```

```
(120, 10, 20, 30)
```

Double-click (or enter) to edit

- Normally `w = w + w` without initializing `w` will generate an error since `w` is not defined
- Within the function, `a = a + a` etc do not generate an error, because `a, b, c` are implicitly assigned values through the arguments
- Note that `int` is immutable, so updating `a, b, c` inside the function has no impact on `x, y, z`

```
def f(a,b,c):
  a.append(5)
  b.append(10)
  c.append(15)
  return(a+b+c)

l1 = [10]
l2 = [20]
l3 = [30]
```

```
10 f(l1,l2,l3), l1, l2, l3
```

```
([10, 5, 20, 10, 30, 15], [10, 5], [20, 10], [30, 15])
```

- Here the arguments are lists, mutable
- `a.append()` etc update the arguments in place, so external `l1, l2, l3` also get updated
- Note that `a.append()` updates `a` in place but returns `None`, see below - be careful not to reassign the list when using `append()` etc

```
 1 def f(a,b,c):
 2   a = a.append(5)
 3   b = b.append(10)
 4   c = c.append(15)
 5   return(a)
 6
 7
 8 l1 = [10]
 9 l2 = [20]
10 l3 = [30]
11 f(l1,l2,l3), l1, l2, l3
```

```
(None, [10, 5], [20, 10], [30, 15])
```

Saved successfully!

```
1 l = [1,2,3]
2 x = l.append(4)
3 x, l
```

```
(None, [1, 2, 3, 4])
```

Functions and parameters

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

```
1 def factorial(n):
2   answer = 1
3   while (n > 0):
4     answer = answer * n
5     n = n - 1
6   return(answer)
7   # Loop computes n * (n-1) * .... * 1
```

```
1 m = 8
2 z = factorial(m)
3 m, z              # Argument m is unaffected by n being modified in
```

```
(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`

```
l = [0]
l2 = l
l.append(1)
l.extend([4,3,6,5])
l3 = sorted(l)
l, l2, l3
```

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

Name spaces

- Variables within functions are local
- But you can use a global immutable value


```
  for i in range(1,n+1):  # list 1,2,3,..,n
    answer = answer * i
  return(answer)
```

```
for i in range(1,10):     # this i does not clash with the i in fact2
  print(fact2(i))
```

```
1
2
6
24
120
720
5040
40320
362880
```

```
def addj(a):
  b = a+j          # j must be defined "globally"
  return(b)
```

```
j = 8
addj(7), j
```

```
(15, 8)
```

```
def addj(a):
  j = 6            # local j, "hides" global j
  b = a+j
```

```
    b = a+j
    return(b)

j = 8
addj(7), j
```

```
(13, 8)
```

```
def addj(a):
    b = a+j          # Generates an error, j undefined.  Update to j
    j = 6            # anywhere in the function makes it a local j.
    return(b)

j = 8
addj(7), j
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-19-9b7f827b902d> in <module>()
      5
      6 j = 8
----> 7 addj(7), j

<ipython-input-19-9b7f827b902d> in addj(a)
      1 def addj(a):
      2     b = a+j          # Generates an error, j undefined.  Update to j
      3     j = 6            # anywhere in the function makes it a local j.
      4     return(b)
      5

UnboundLocalError: local variable 'j' referenced before assignment
```


Strings

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

```
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
```

```
('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

```
l = [0,1,2,3,4]
y[4][0][0], x[2:4], x+' '+b, x[-10:10], y[0]+y[1:]
[l[0]]+l[1:]
```

```
[0, 1, 2, 3, 4]
```

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

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

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

Saved successfully!

```
l5]
l
```

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

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

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-640e2077f0c3> in <module>()
      1 h = 'hello'
----> 2 h[3:5] = 'ps'

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

SEARCH STACK OVERFLOW

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

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

```
'helps'
```

Basic input and output

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

```
1 x = input()
```

```
45
```

```
1 x
```

```
'45'
```

```
1 x + 52
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-e17699fa9569> in <module>()
----> 1 x + 52

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

Saved successfully!

Type conversion

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

```
1 int(x) + 52
```

```
97
```

- Any type name can be used as a type converter

```
1 list("hello")
```

```
['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

```
1 int("5.2")
```

```
---------------------------------------------------------------------------
```

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

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

```
1 int("a7",16)
```

```
167
```

Output

- `print(x1,x2,...,xn)`
- Implicitly each `xi` is converted to `str(xi)`

```
1 x = 'a'
2 print(7,x)
```

```
7 a
```

Tuples

Saved successfully!

t square

t)

- Otherwise manipulate using indices, slices etc

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

```
(1, (3, 4), 3)
```

Multiple assignment using tuples

```
1 x,y = 3,5  # same as (x,y) = (3,5)
```

```
1 y, x
```

```
(5, 3)
```

Useful to initialize many things at the start of a function

```
1 i,j,k,factorlist = 0,0,0,[]
2 i,j,k,factorlist
```

```
(0, 0, 0, [])
```

Exchange the values of two variables

- 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)
```

```
1 x,y = 77,55
2 x,y = y,x
3 x,y
```

```
(55, 77)
```

Saved successfully!