PDSP 2026, Lecture 07, 1 September 2026

Data types in Python

- Extracting unique elements from a list using a dictionary

In [1]:
```
def uniqd(l):
    for x in l:
        uniqdict[x] = 1
    return(list(uniqdict))
```

In [2]:
```
uniqd([1,2,4,3,2,4,1])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 uniqd([1,2,4,3,2,4,1])

Cell In[1], line 3, in uniqd(l)
      1 def uniqd(l):
      2     for x in l:
----> 3         uniqdict[x] = 1
      4     return(list(uniqdict))

NameError: name 'uniqdict' is not defined
```

- For `uniqdict[x] = 1` to make sense, Python needs to know that `uniqdict` is a dictionary
- It is important to initialize `uniqdict` to `{}`

In [3]:
```
def uniqd(l):
    uniqdict = {}
    for x in l:
        uniqdict[x] = 1
    return(list(uniqdict))
```

In [4]:
```
uniqd([1,2,4,3,2,4,1])
```

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

- Likewise for a function that builds up a list

In [5]:
```
def factors(n):
    for i in range(1,n+1):
        if (n%i == 0):
            factorlist.append(i)
    return(factorlist)
```

In [6]:
```
factors(24)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 factors(24)

Cell In[5], line 4, in factors(n)
      1 def factors(n):
      2     for i in range(1,n+1):
      3         if (n%i == 0):
----> 4             factorlist.append(i)
      5     return(factorlist)

NameError: name 'factorlist' is not defined
```

In [7]:
```
def factors(n):
    factorlist = []
    for i in range(1,n+1):
        if (n%i == 0):
            factorlist.append(i)
    return(factorlist)
```

In [8]:
```
factors(24)
```

Out[8]:
```
[1, 2, 3, 4, 6, 8, 12, 24]
```

Nested collections

- List of lists, list of tuples, dictionary whose values are lists ...
- `matchlist` is a list of tuples
- Use two indices to extract a value
  - `matchlist[2]` is `("Toronto","CAN","BIH",1,1,0,0,1.35,0.98)`
  - `matchlist[2][1]` is `"CAN"`

In [9]:
```
matchlist = [
  ("Mexico City","MEX","RSA",2,0,0,0,1.84,0.52),
  ("Zapopan","KOR","CZE",2,1,0,0,1.45,1.12),
  ("Toronto","CAN","BIH",1,1,0,0,1.35,0.98),
  ("Inglewood","USA","PAR",4,1,0,0,2.76,0.88),
  ("Foxborough","QAT","SUI",1,1,0,0,0.78,1.54),
  ("East Rutherford","BRA","MAR",1,1,0,0,1.62,1.1),
  ("Philadelphia","HAI","SCO",0,1,0,0,1.25,0.65),
  ("Seattle","AUS","TUR",2,0,0,0,1.48,0.72),
  ("Atlanta","GER","CUW",7,1,0,0,4.82,0.44),
  ("Santa Clara","NED","JPN",2,2,0,0,1.95,1.82),
  ("Houston","CIV","ECU",1,0,0,0,1.1,0.95),
  ("Kansas City","SWE","TUN",5,1,0,0,3.12,0.78),
  ("Miami Gardens","ESP","CPV",0,0,0,0,2.15,0.35),
  ("Arlington","BEL","EGY",1,1,0,0,1.58,1.12),
  ("Guadalupe","KSA","URU",1,1,0,0,0.85,1.76),
  ("Vancouver","IRN","NZL",2,2,0,0,1.88,1.2),
  ("East Rutherford","FRA","SEN",3,1,0,0,2.44,1.05),
  ("Foxborough","IRQ","NOR",1,4,0,0,0.68,2.85),
```

```
("Inglewood","ARG","ALG",3,0,0,0,2.65,0.42),
("Santa Clara","AUT","JOR",3,1,0,0,1.98,0.88),
("Atlanta","POR","COD",1,1,0,0,1.78,1.15),
("Miami Gardens","ENG","CRO",4,2,0,0,2.92,1.64),
("Houston","GHA","PAN",1,0,0,0,1.3,0.75),
("Seattle","UZB","COL",1,3,0,0,0.95,2.1),
("Atlanta","MEX","KOR",1,0,0,0,1.62,0.58),
("Inglewood","CZE","RSA",1,1,0,0,1.35,1.15),
("Vancouver","CAN","QAT",6,0,0,0,3.82,0.22),
("Mexico City","SUI","BIH",4,1,0,0,2.45,0.88),
("East Rutherford","BRA","HAI",3,0,0,0,1.16,0.9),
("Arlington","SCO","MAR",0,1,0,0,0.97,0.54),
("Toronto","USA","AUS",2,0,0,0,1.21,0.32),
("Guadalupe","TUR","PAR",0,1,0,0,0.4,0.3),
("Foxborough","GER","CIV",2,1,0,0,1.85,0.72),
("Kansas City","ECU","CUW",0,0,0,0,1.92,0.08),
("Philadelphia","NED","SWE",5,1,0,0,3.45,0.88),
("Seattle","TUN","JPN",0,4,0,0,0.35,3.12),
("Houston","BEL","IRN",0,0,0,0,1.45,0.62),
("Miami Gardens","NZL","EGY",1,3,0,0,0.55,2.15),
("Santa Clara","ESP","KSA",4,0,0,0,3.28,0.25),
("East Rutherford","URU","CPV",2,2,0,0,1.65,1.1),
("Arlington","FRA","IRQ",3,0,0,0,2.21,0.45),
("Toronto","NOR","SEN",3,2,0,0,1.95,1.48),
("Guadalupe","ARG","AUT",2,0,0,0,1.78,0.55),
("Foxborough","JOR","ALG",1,2,0,0,0.88,1.92),
("Kansas City","POR","UZB",5,0,0,0,3.54,0.42),
("Philadelphia","COL","COD",1,0,0,0,1.62,0.58),
("Seattle","ENG","GHA",0,0,0,0,1.12,0.89),
("Mexico City","PAN","CRO",0,1,0,0,0.42,1.48),
("Mexico City","CZE","MEX",0,3,0,0,1.1,1.27),
("Zapopan","RSA","KOR",1,0,0,0,1.26,1.39),
("Vancouver","SUI","CAN",2,1,0,0,0.71,1.35),
("Toronto","BIH","QAT",3,1,0,0,1.48,0.95),
("East Rutherford","SCO","BRA",0,3,0,0,0.45,2.18),
("Foxborough","MAR","HAI",4,2,0,0,2.35,1.12),
("Inglewood","TUR","USA",3,2,0,0,1.62,1.45),
("Santa Clara","PAR","AUS",0,0,0,0,0.55,0.48),
("Atlanta","ECU","GER",2,1,0,0,1.28,1.55),
("Houston","CUW","CIV",0,2,0,0,0.32,1.68),
("Kansas City","TUN","NED",1,3,0,0,0.62,2.15),
("Seattle","JPN","SWE",1,1,0,0,1.12,0.95),
("Arlington","NZL","BEL",1,5,0,0,0.48,3.25),
("Guadalupe","EGY","IRN",1,1,0,0,0.85,0.92),
("Miami Gardens","URU","ESP",0,1,0,0,0.78,1.35),
("Philadelphia","CPV","KSA",0,0,0,0,0.42,0.38),
("Foxborough","NOR","FRA",1,4,0,0,1.3,0.96),
("East Rutherford","SEN","IRQ",5,0,0,0,3.01,0.14),
("Santa Clara","JOR","ARG",1,3,0,0,0.76,2.13),
("Inglewood","ALG","AUT",3,3,0,0,1.62,1.44),
("Miami Gardens","COL","POR",0,0,0,0,1.62,0.73),
("Atlanta","COD","UZB",3,1,0,0,2.35,0.2),
("East Rutherford","PAN","ENG",0,2,0,0,0.69,1.39),
("Philadelphia","CRO","GHA",2,1,0,0,0.42,0.74),
("Inglewood","RSA","CAN",0,1,0,0,0.13,1.32),
("Houston","BRA","JPN",2,1,0,0,1.69,0.23),
("Foxborough","GER","PAR",1,1,3,4,0.72,0.42),
("Guadalupe","NED","MAR",1,1,2,3,0.23,1.4),
("Arlington","CIV","NOR",1,2,0,0,1.15,2.02),
("East Rutherford","FRA","SWE",3,0,0,0,3.17,0.67),
```

```
  ("Mexico City","MEX","ECU",2,0,0,0,1.02,0.73),
  ("Atlanta","ENG","COD",2,1,0,0,2.04,0.76),
  ("Seattle","BEL","SEN",3,2,0,0,1.74,3.58),
  ("Santa Clara","USA","BIH",2,0,0,0,0.92,0.25),
  ("Inglewood","ESP","AUT",3,0,0,0,2.84,0.32),
  ("Toronto","POR","CRO",2,1,0,0,2.18,1.34),
  ("Vancouver","SUI","ALG",2,0,0,0,2.52,0.73),
  ("Arlington","AUS","EGY",1,1,2,4,0.87,1.36),
  ("Miami Gardens","ARG","CPV",3,2,0,0,2.16,0.46),
  ("Kansas City","COL","GHA",1,0,0,0,2.19,0.26),
  ("Philadelphia","PAR","FRA",0,1,0,0,0.3,1.8),
  ("Houston","CAN","MAR",0,3,0,0,0.8,1.2),
  ("Arlington","BRA","NOR",1,2,0,0,2.61,1.05),
  ("Mexico City","MEX","ENG",2,3,0,0,1.88,1.61),
  ("Miami Gardens","POR","ESP",0,1,0,0,0.63,1.69),
  ("Santa Clara","USA","BEL",1,4,0,0,0.67,2.15),
  ("Kansas City","ARG","EGY",3,2,0,0,2.12,1.54),
  ("Seattle","SUI","COL",0,0,4,3,1.15,1.46),
  ("Foxborough","FRA","MAR",2,0,0,0,3.1,0.13),
  ("Inglewood","ESP","BEL",2,1,0,0,2.08,0.37),
  ("Miami Gardens","NOR","ENG",1,2,0,0,0.77,0.96),
  ("Kansas City","ARG","SUI",3,1,0,0,2.0,0.53),
  ("Arlington","FRA","ESP",0,2,0,0,0.3,1.63),
  ("Atlanta","ENG","ARG",1,2,0,0,0.54,1.80),
  ("Miami Gardens","FRA","ENG",4,6,0,0,2.88,2.88),
  ("East Rutherford","ESP","ARG",1,0,0,0,0.52,0.09)
]
```

List of venues where each team played in FIFA 2026

- Initially, build a dictionary of dictionaries
  - Outer keys are team names
  - Inner dictionary keys are venues where team played
- Finally, convert dictionary of dictionaries into dictionary of lists

In [10]:
```
def get_venues(l):
    venues = {}
    for m in l:
        venue,team1,team2 = m[0],m[1],m[2]
        for t in team1,team2:
            if t in venues:
                venues[t][venue] = 1   # Create/update entry for current
            else:
                venues[t] = {venue:1}  # Create a dictionary for venues[t
                                       # Equivalent to
                                       #   venues[t] = {}
                                       #   venues[t[venue] = 1
    venuelists = {}
    for t in sorted(venues):           # Scan teams in alphabetical order
        venuelists[t] = list(venues[t]) # Venues will be listed in the or
    return(venuelists)
```

In [11]:
```
get_venues(matchlist)
```

Out[11]:
```
{'ALG': ['Inglewood', 'Foxborough', 'Vancouver'],
 'ARG': ['Inglewood',
  'Guadalupe',
  'Santa Clara',
  'Miami Gardens',
  'Kansas City',
  'Atlanta',
  'East Rutherford'],
 'AUS': ['Seattle', 'Toronto', 'Santa Clara', 'Arlington'],
 'AUT': ['Santa Clara', 'Guadalupe', 'Inglewood'],
 'BEL': ['Arlington', 'Houston', 'Seattle', 'Santa Clara', 'Inglewood'],
 'BIH': ['Toronto', 'Mexico City', 'Santa Clara'],
 'BRA': ['East Rutherford', 'Houston', 'Arlington'],
 'CAN': ['Toronto', 'Vancouver', 'Inglewood', 'Houston'],
 'CIV': ['Houston', 'Foxborough', 'Arlington'],
 'COD': ['Atlanta', 'Philadelphia'],
 'COL': ['Seattle', 'Philadelphia', 'Miami Gardens', 'Kansas City'],
 'CPV': ['Miami Gardens', 'East Rutherford', 'Philadelphia'],
 'CRO': ['Miami Gardens', 'Mexico City', 'Philadelphia', 'Toronto'],
 'CUW': ['Atlanta', 'Kansas City', 'Houston'],
 'CZE': ['Zapopan', 'Inglewood', 'Mexico City'],
 'ECU': ['Houston', 'Kansas City', 'Atlanta', 'Mexico City'],
 'EGY': ['Arlington', 'Miami Gardens', 'Guadalupe', 'Kansas City'],
 'ENG': ['Miami Gardens',
  'Seattle',
  'East Rutherford',
  'Atlanta',
  'Mexico City'],
 'ESP': ['Miami Gardens',
  'Santa Clara',
  'Inglewood',
  'Arlington',
  'East Rutherford'],
 'FRA': ['East Rutherford',
  'Arlington',
  'Foxborough',
  'Philadelphia',
  'Miami Gardens'],
 'GER': ['Atlanta', 'Foxborough'],
 'GHA': ['Houston', 'Seattle', 'Philadelphia', 'Kansas City'],
 'HAI': ['Philadelphia', 'East Rutherford', 'Foxborough'],
 'IRN': ['Vancouver', 'Houston', 'Guadalupe'],
 'IRQ': ['Foxborough', 'Arlington', 'East Rutherford'],
 'JOR': ['Santa Clara', 'Foxborough'],
 'JPN': ['Santa Clara', 'Seattle', 'Houston'],
 'KOR': ['Zapopan', 'Atlanta'],
 'KSA': ['Guadalupe', 'Santa Clara', 'Philadelphia'],
 'MAR': ['East Rutherford', 'Arlington', 'Foxborough', 'Guadalupe', 'Hou
ston'],
 'MEX': ['Mexico City', 'Atlanta'],
 'NED': ['Santa Clara', 'Philadelphia', 'Kansas City', 'Guadalupe'],
 'NOR': ['Foxborough', 'Toronto', 'Arlington', 'Miami Gardens'],
 'NZL': ['Vancouver', 'Miami Gardens', 'Arlington'],
 'PAN': ['Houston', 'Mexico City', 'East Rutherford'],
 'PAR': ['Inglewood',
  'Guadalupe',
  'Santa Clara',
  'Foxborough',
  'Philadelphia'],
 'POR': ['Atlanta', 'Kansas City', 'Miami Gardens', 'Toronto'],
```

```
 'QAT': ['Foxborough', 'Vancouver', 'Toronto'],
 'RSA': ['Mexico City', 'Inglewood', 'Zapopan'],
 'SCO': ['Philadelphia', 'Arlington', 'East Rutherford'],
 'SEN': ['East Rutherford', 'Toronto', 'Seattle'],
 'SUI': ['Foxborough', 'Mexico City', 'Vancouver', 'Seattle', 'Kansas Ci
ty'],
 'SWE': ['Kansas City', 'Philadelphia', 'Seattle', 'East Rutherford'],
 'TUN': ['Kansas City', 'Seattle'],
 'TUR': ['Seattle', 'Guadalupe', 'Inglewood'],
 'URU': ['Guadalupe', 'East Rutherford', 'Miami Gardens'],
 'USA': ['Inglewood', 'Toronto', 'Santa Clara'],
 'UZB': ['Seattle', 'Kansas City', 'Atlanta']}
```

In [12]:
```
def get_venues(l):
    venues = {}
    for m in l:
        venue,team1,team2 = m[0],m[1],m[2]
        for t in team1,team2:
            if t in venues:
                venues[t][venue] = 1   # Create/update entry for current
            else:
                venues[t] = {venue:1}  # Create a dictionary for venues[t
                                       # Equivalent to
                                       #   venues[t] = {}
                                       #   venues[t[[venue] = 1
    venuelists = {}
    for t in sorted(venues):           # Scan teams in alphabetical order
        venuelists[t] = sorted(list(venues[t]))  # Sort venues in alphabe
    return(venuelists)
```

In [13]:
```
get_venues(matchlist)
```

Out[13]:
```
{'ALG': ['Foxborough', 'Inglewood', 'Vancouver'],
 'ARG': ['Atlanta',
  'East Rutherford',
  'Guadalupe',
  'Inglewood',
  'Kansas City',
  'Miami Gardens',
  'Santa Clara'],
 'AUS': ['Arlington', 'Santa Clara', 'Seattle', 'Toronto'],
 'AUT': ['Guadalupe', 'Inglewood', 'Santa Clara'],
 'BEL': ['Arlington', 'Houston', 'Inglewood', 'Santa Clara', 'Seattle'],
 'BIH': ['Mexico City', 'Santa Clara', 'Toronto'],
 'BRA': ['Arlington', 'East Rutherford', 'Houston'],
 'CAN': ['Houston', 'Inglewood', 'Toronto', 'Vancouver'],
 'CIV': ['Arlington', 'Foxborough', 'Houston'],
 'COD': ['Atlanta', 'Philadelphia'],
 'COL': ['Kansas City', 'Miami Gardens', 'Philadelphia', 'Seattle'],
 'CPV': ['East Rutherford', 'Miami Gardens', 'Philadelphia'],
 'CRO': ['Mexico City', 'Miami Gardens', 'Philadelphia', 'Toronto'],
 'CUW': ['Atlanta', 'Houston', 'Kansas City'],
 'CZE': ['Inglewood', 'Mexico City', 'Zapopan'],
 'ECU': ['Atlanta', 'Houston', 'Kansas City', 'Mexico City'],
 'EGY': ['Arlington', 'Guadalupe', 'Kansas City', 'Miami Gardens'],
 'ENG': ['Atlanta',
  'East Rutherford',
  'Mexico City',
  'Miami Gardens',
  'Seattle'],
 'ESP': ['Arlington',
  'East Rutherford',
  'Inglewood',
  'Miami Gardens',
  'Santa Clara'],
 'FRA': ['Arlington',
  'East Rutherford',
  'Foxborough',
  'Miami Gardens',
  'Philadelphia'],
 'GER': ['Atlanta', 'Foxborough'],
 'GHA': ['Houston', 'Kansas City', 'Philadelphia', 'Seattle'],
 'HAI': ['East Rutherford', 'Foxborough', 'Philadelphia'],
 'IRN': ['Guadalupe', 'Houston', 'Vancouver'],
 'IRQ': ['Arlington', 'East Rutherford', 'Foxborough'],
 'JOR': ['Foxborough', 'Santa Clara'],
 'JPN': ['Houston', 'Santa Clara', 'Seattle'],
 'KOR': ['Atlanta', 'Zapopan'],
 'KSA': ['Guadalupe', 'Philadelphia', 'Santa Clara'],
 'MAR': ['Arlington', 'East Rutherford', 'Foxborough', 'Guadalupe', 'Hou
ston'],
 'MEX': ['Atlanta', 'Mexico City'],
 'NED': ['Guadalupe', 'Kansas City', 'Philadelphia', 'Santa Clara'],
 'NOR': ['Arlington', 'Foxborough', 'Miami Gardens', 'Toronto'],
 'NZL': ['Arlington', 'Miami Gardens', 'Vancouver'],
 'PAN': ['East Rutherford', 'Houston', 'Mexico City'],
 'PAR': ['Foxborough',
  'Guadalupe',
  'Inglewood',
  'Philadelphia',
  'Santa Clara'],
 'POR': ['Atlanta', 'Kansas City', 'Miami Gardens', 'Toronto'],
```

```
 'QAT': ['Foxborough', 'Toronto', 'Vancouver'],
 'RSA': ['Inglewood', 'Mexico City', 'Zapopan'],
 'SCO': ['Arlington', 'East Rutherford', 'Philadelphia'],
 'SEN': ['East Rutherford', 'Seattle', 'Toronto'],
 'SUI': ['Foxborough', 'Kansas City', 'Mexico City', 'Seattle', 'Vancouv
er'],
 'SWE': ['East Rutherford', 'Kansas City', 'Philadelphia', 'Seattle'],
 'TUN': ['Kansas City', 'Seattle'],
 'TUR': ['Guadalupe', 'Inglewood', 'Seattle'],
 'URU': ['East Rutherford', 'Guadalupe', 'Miami Gardens'],
 'USA': ['Inglewood', 'Santa Clara', 'Toronto'],
 'UZB': ['Atlanta', 'Kansas City', 'Seattle']}
```

Matrices

- Matrix is a list of lists
- Each row is a list of values `[v1,v2,...vm]`
- Matrix is a list of rows `[ row1, row2, ..., rown ]`

Transpose a matrix

- Columns of original matrix are rows of transpose
- If input is square, can swap `M[i][j]` and `M[j][i]`
- If not square, need to construct transpose of correct dimension

In [14]:
```
def transpose(mat):
    # Extract dimensions of input -- assume well-formed, non-empty
    nrows = len(mat)
    ncols = len(mat[0])
    # Create an empty matrix for the transpose
    trmat = []
    for i in range(ncols):
        trmat.append([])

    # Populate the transpose -- append each M[i][j] to row j of transpose
    for i in range(nrows):
        for j in range(ncols):
            trmat[j].append(mat[i][j])
    return(trmat)
```

In [15]:
```
m = [[0,3],[1,4],[2,5]]
mt = transpose(m)
```

In [16]:
```
m, mt
```

Out[16]:
```
([[0, 3], [1, 4], [2, 5]], [[0, 1, 2], [3, 4, 5]])
```

- Multiplication is repeated addition
- `[0] * 5` is `[0]+[0]+[0]+[0]+[0]`

In [17]:
```
[0]*5
```

Out[17]:
```
[0, 0, 0, 0, 0]
```

- Create an $n \times n$ matrix of zeros

In [18]:
```
def make_zero(n):
    zero_row = [0] * n  # Create one row of zeros
    zero_matrix = []      # Empty identity matrix
    for i in range(n):
        zero_matrix.append(zero_row)
    return(zero_matrix)
```

In [19]:
```
make_zero(5)
```

Out[19]:
```
[[0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]
```

- Create an $n \times n$ identity matrix

In [20]:
```
def make_identity(n):
    identity_matrix = make_zero(n)
    for i in range(n):
        identity_matrix[i][i] = 1
    return(identity_matrix)
```

In [21]:
```
make_identity(5)
```

Out[21]:
```
[[1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]]
```

- What happened?

In [22]:
```
l1 = [4,3,1,2]
```

In [23]:
```
l2 = l1
```

In [24]:
```
l2.sort()
```

In [25]:
```
l1, l2
```

Out[25]:
```
([1, 2, 3, 4], [1, 2, 3, 4])
```

Mutable and immutable values

- Lists and dictionaries can be updated in place
  - Can reassign `l[i]` or `d[k]`
  - These are *mutable* values
- Numbers (`int`, `float`), booleans, strings, tuples cannot be updated in place
  - Immutable values

Mutability and assignment

- Assigning a mutable value creates an *alias*
- Updating through either the old or the new name indirectly affects the other

In [26]:
```
l = [1,2,3]
newl = l
newl[0] = 4
```

In [27]:
```
l, newl
```

Out[27]:
```
([4, 2, 3], [4, 2, 3])
```

In [28]:
```
l[1] = 5
```

In [29]:
```
l, newl
```

Out[29]:
```
([4, 5, 3], [4, 5, 3])
```

- For immutable values, assignment behaves as we would expect
- The two names can be updated without affecting each other
  - It is as though assignment copies the value

In [30]:
```
x = 17
y = x
y = 19
```

In [31]:
```
x, y
```

Out[31]:
```
(17, 19)
```

In [32]:
```
x = 18
```

In [33]:
```
x, y
```

Out[33]:
```
(18, 19)
```

Mutable and immutable values

- Lists and dictionaries are mutable
- `int`, `float`, `bool`, strings, tuples are immutable

Slices and copying lists

- A slice creates a new list
- `l[0:len(l)]` is a faithful copy of `l`
  - Abbreviate as `l[:]`, *full slice*
- Assigning a full slice makes a disjoint copy of a list

In [34]:
```
l1 = [1,2,3]
l2 = l1[:]
```

In [35]:
```
l1,l2
```

Out[35]:
```
([1, 2, 3], [1, 2, 3])
```

In [36]:
```
l1[2] = 6
l2[0] = 4
```

In [37]:
```
l1, l2
```

Out[37]:
```
([1, 2, 6], [4, 2, 3])
```

Calling functions

- Suppose we have a function definition `def f(a,b):` and a function call `f(x,y)`
- When `f(x,y)` is executed, it is as though we start `f` with the assignments

```
a = x
b = y
```

- This explains how/when values can be updated within a function

In [38]:
```
def factorial(n):
    ans = 1
    while n >= 1:
        ans = ans * n
        n = n-1
    return(ans)
```

In [39]:
```
x = 6
y = factorial(x)
```

In [40]:
```
x,y
```

Out[40]:
```
(6, 720)
```

- Inside the function, the parameter `n` is decremented to `0`
- `n` is derived from the variable `x` passed when the function is called
- Since `x` is immutable, the implicit assignment `n = x` *copies* the value of `x` into `n`

- Updating `n` has no effect on `x`

- This also means we cannot write a function `swap` along the following lines

In [41]:
```
def swap(x,y):
    (x,y) = (y,x)
    return
```

In [42]:
```
m = 5
n = 7
swap(m,n)
```

In [43]:
```
m,n
```

Out[43]: (5, 7)

- This will not work with mutable values either
- The problem is the reassignment inside the function

In [44]:
```
list1 = [1,2,3]
list2 = [4,5,6]
swap(list1,list2)
```

In [45]:
```
list1, list2
```

Out[45]: ([1, 2, 3], [4, 5, 6])

Passing mutable values to a function

- Passing an argument is like executing an assignment statement before starting the function
- For mutable values, this aliases the function parameter to the called value
- In place changes in the function affect the value outside the function

In [46]:
```
def concat(l1,l2):
    l1.extend(l2)
    return
```

In [47]:
```
l3 = [1,2,3]
l4 = [4,5,6]
concat(l3,l4)
```

In [48]:
```
l3,l4
```

Out[48]: ([1, 2, 3, 4, 5, 6], [4, 5, 6])

- If we pass a slice, the value in the function is a disjoint copy

In [49]:
```
l3 = [1,2,3]
l4 = [4,5,6]
concat(l3[:],l4[:])
```

In [50]:
```
l3,l4
```

Out[50]:
```
([1, 2, 3], [4, 5, 6])
```

- However, reassigning the variable inside the function creates a new value not connected to the outer value

In [51]:
```
def concat2(l1,l2):
    l1 = l1 + l2
    return
```

In [52]:
```
l3 = [1,2,3]
l4 = [4,5,6]
concat2(l3,l4)
```

In [53]:
```
l3,l4  # No effect - reassignment in function creates a local copy
```

Out[53]:
```
([1, 2, 3], [4, 5, 6])
```

- In fact, our problem with `swap()` applies to mutable values as well
- The statement `(m,n) = (n,m)` is a reassignment and creates new values inside the function

In [54]:
```
swap(l3,l4)
```

In [55]:
```
l3,l4
```

Out[55]:
```
([1, 2, 3], [4, 5, 6])
```

- Be careful not to mix reassignment with in-place modification
- What is the outcome of the following?

In [56]:
```
def myappend(l,x):
    l = l.append(x)
    return(l)
```

In [57]:
```
l1 = [1,2]
l1 = myappend(l1,3)
```

In [58]:
```
l1
```

In [59]:
```
print(l1)
```

```
None
```

- `None` is a special value in Python that explicitly represents that no value is assigned

- A function that does not return a value returns `None`
- In the notebook, the value is "empty", but `print()` displays it as `None`
  - In other words, `str(None)` converts the value `None` to the string `"None"`
- `None` has its own type which is not compatible with any other type, so no operations are legal

In [60]:
```
str(None)
```

Out[60]:
```
'None'
```

In [61]:
```
print(None)
```

```
None
```

In [62]:
```
type(None)
```

Out[62]:
```
NoneType
```

- Setting a variable to `None` is different from leaving it undefined

In [63]:
```
x = 7
```

In [64]:
```
type(x)
```

Out[64]:
```
int
```

In [65]:
```
del(x)
```

In [66]:
```
x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[66], line 1
----> 1 x

NameError: name 'x' is not defined
```

In [67]:
```
x = None
```

In [68]:
```
x
```

- We can test if a variable is set to `None`
- We will use this later

In [69]:
```
x == None
```

Out[69]:
```
True
```