PDSP 2026, Lecture 09, 8 September 2026

In [1]:
```
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)
]
```

In [2]:

```
def expand(s):
    teamdict = {
        'ALG':'Algeria',
        'ARG':'Argentina',
        'AUS':'Australia',
        'AUT':'Austria',
        'BEL':'Belgium',
        'BIH':'Bosnia Herzegovina',
        'BRA':'Brazil',
        'CAN':'Canada',
        'CIV':'Ivory Coast',
        'COD':'Congo',
        'COL':'Colombia',
        'CPV':'Cape Verde',
        'CRO':'Croatia',
        'CUW':'Curacao',
        'CZE':'Czech Republic',
        'ECU':'Ecuador',
```

```
        'EGY':'Egypt',
        'ENG':'England',
        'ESP':'Spain',
        'FRA':'France',
        'GER':'Germany',
        'GHA':'Ghana',
        'HAI':'Haiti',
        'IRN':'Iran',
        'IRQ':'Iraq',
        'JOR':'Jordan',
        'JPN':'Japan',
        'KOR':'Korea',
        'KSA':'Saudi Arabia',
        'MAR':'Morocco',
        'MEX':'Mexico',
        'NED':'Netherlands',
        'NOR':'Norway',
        'NZL':'New Zealand',
        'PAN':'Panama',
        'PAR':'Paraguay',
        'POR':'Portugal',
        'QAT':'Qatar',
        'RSA':'South Africa',
        'SCO':'Scotland',
        'SEN':'Senegal',
        'SUI':'Switzerland',
        'SWE':'Sweden',
        'TUN':'Tunisia',
        'TUR':'Turkey',
        'URU':'Uruguay',
        'USA':'United States of America',
        'UZB':'Uzbekistan'}
    if (s in teamdict):
        return(teamdict[s])
    else:
        return('No info')
```

In [3]:
```
def get_teams(l):
    teamdict = {}
    for m in l:
        team1,team2 = m[1],m[2]
        teamdict[team1] = 1
        teamdict[team2] = 2
    return(sorted(list(teamdict)))
```

Map, filter, list comprehension

Map

- Apply a function $f()$ to each element in a list
- Convert $[x_0, x_1, \ldots, x_{n-1}]$ to $[f(x_0), f(x_1), \ldots, f(x_{n-1})]$
- In Python, `map(f,l)` applies `f` to each element of `l`

In [4]:
```
def square(m):
    return m*m
```

In [5]:
```
list(map(square,range(1,7)))
```

Out[5]:
```
[1, 4, 9, 16, 25, 36]
```

Filter

- Extract items from a list that satisfy a property
  - For instance, extract the even numbers from a list of numbers
- A *property* is a function $p()$ that maps each input to `True` or `False
  - Check if each item x in a list satisfies a property $p(x)$
  - Retain only such elements
  - Filters out elements that do not satisfy $p()$
- In Python, `filter(p,l)`

In [6]:
```
def even(m):
    return m%2 == 0
```

In [7]:
```
list(filter(even,range(1,7)))
```

Out[7]:
```
[2, 4, 6]
```

- Common to combine `map` and `filter`
- Squares of even numbers in `[1,2,...,6]`

In [8]:
```
list(map(square,(filter(even,range(1,7)))))
```

Out[8]:
```
[4, 16, 36]
```

List comprehension

- Combine map and filter to create a list
- *Set comprehension*: Squares of positive even integers = $\{x^2 \mid x \in \mathbb{Z}, x > 0, x \bmod 2 = 0\}$
- In Python: `[ f(x) for x in l if p(x) ]`

- Using list comprehension to create a zero matrix

- First a row of `n` zeros

In [9]:
```
def zerorow(n):
    return [0 for i in range(n)]
```

In [10]:
```
zerorow(5)
```

Out[10]:
```
[0, 0, 0, 0, 0]
```

- Now, create a matrix with `n` distinct copies of `zerorow

In [11]:
```
def zeromat(n):
    return [zerorow(n) for i in range(n)]
```

In [12]:
```
zeromat(5)
```

Out[12]:
```
[[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]]
```

- Changing a diagonal entry does not affect the other rows

In [13]:
```
zm = zeromat(5)
zm[2][2] = 1
```

In [14]:
```
zm
```

Out[14]:
```
[[0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]
```

- Create an identity matrix

In [15]:
```
def idmat(n):
    zmat = zeromat(n)
    for i in range(n):
        zmat[i][i] = 1
    return zmat
```

In [16]:
```
idmat(5)
```

Out[16]:
```
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 1]]
```

Example

- Full names of all teams in FIFA 2026

In [17]:
```
[ expand(t) for t in get_teams(matchlist) ]
```

Out[17]:
```
['Algeria',
 'Argentina',
 'Australia',
 'Austria',
 'Belgium',
 'Bosnia Herzegovina',
 'Brazil',
 'Canada',
 'Ivory Coast',
 'Congo',
 'Colombia',
 'Cape Verde',
 'Croatia',
 'Curacao',
 'Czech Republic',
 'Ecuador',
 'Egypt',
 'England',
 'Spain',
 'France',
 'Germany',
 'Ghana',
 'Haiti',
 'Iran',
 'Iraq',
 'Jordan',
 'Japan',
 'Korea',
 'Saudi Arabia',
 'Morocco',
 'Mexico',
 'Netherlands',
 'Norway',
 'New Zealand',
 'Panama',
 'Paraguay',
 'Portugal',
 'Qatar',
 'South Africa',
 'Scotland',
 'Senegal',
 'Switzerland',
 'Sweden',
 'Tunisia',
 'Turkey',
 'Uruguay',
 'United States of America',
 'Uzbekistan']
```

- List both teams in matches where MEX was home team

- Equivalent of relational algebra selection and projection
- Project matchlist onto columns `team 1, team 2` (columns 1,2) where MEX was home team
  - Filter by MEX being home team
  - Project onto columns 1,2

In [18]:
```
[ m for m in matchlist if m[1] == 'MEX' ]
```

Out[18]:
```
[('Mexico City', 'MEX', 'RSA', 2, 0, 0, 0, 1.84, 0.52),
 ('Atlanta', 'MEX', 'KOR', 1, 0, 0, 0, 1.62, 0.58),
 ('Mexico City', 'MEX', 'ECU', 2, 0, 0, 0, 1.02, 0.73),
 ('Mexico City', 'MEX', 'ENG', 2, 3, 0, 0, 1.88, 1.61)]
```

In [19]:
```
[ (m[1],m[2]) for m in matchlist if m[1] == 'MEX' ]
```

Out[19]:
```
[('MEX', 'RSA'), ('MEX', 'KOR'), ('MEX', 'ECU'), ('MEX', 'ENG')]
```

- Same, with full names

In [20]:
```
[ (expand(m[1]),expand(m[2])) for m in matchlist if m[1] == 'MEX' ]
```

Out[20]:
```
[('Mexico', 'South Africa'),
 ('Mexico', 'Korea'),
 ('Mexico', 'Ecuador'),
 ('Mexico', 'England')]
```

- List comprehension notation can also be used to create dictionaries
- Create a dictionary matching team abbreviations to full names for teams in FIFA 2026

In [21]:
```
newd = { t:expand(t) for t in get_teams(matchlist) }
```

In [22]:
```
newd
```

Out[22]:
```
{'ALG': 'Algeria',
 'ARG': 'Argentina',
 'AUS': 'Australia',
 'AUT': 'Austria',
 'BEL': 'Belgium',
 'BIH': 'Bosnia Herzegovina',
 'BRA': 'Brazil',
 'CAN': 'Canada',
 'CIV': 'Ivory Coast',
 'COD': 'Congo',
 'COL': 'Colombia',
 'CPV': 'Cape Verde',
 'CRO': 'Croatia',
 'CUW': 'Curacao',
 'CZE': 'Czech Republic',
 'ECU': 'Ecuador',
 'EGY': 'Egypt',
 'ENG': 'England',
 'ESP': 'Spain',
 'FRA': 'France',
 'GER': 'Germany',
 'GHA': 'Ghana',
 'HAI': 'Haiti',
 'IRN': 'Iran',
 'IRQ': 'Iraq',
 'JOR': 'Jordan',
 'JPN': 'Japan',
 'KOR': 'Korea',
 'KSA': 'Saudi Arabia',
 'MAR': 'Morocco',
 'MEX': 'Mexico',
 'NED': 'Netherlands',
 'NOR': 'Norway',
 'NZL': 'New Zealand',
 'PAN': 'Panama',
 'PAR': 'Paraguay',
 'POR': 'Portugal',
 'QAT': 'Qatar',
 'RSA': 'South Africa',
 'SCO': 'Scotland',
 'SEN': 'Senegal',
 'SUI': 'Switzerland',
 'SWE': 'Sweden',
 'TUN': 'Tunisia',
 'TUR': 'Turkey',
 'URU': 'Uruguay',
 'USA': 'United States of America',
 'UZB': 'Uzbekistan'}
```

- Recall that `uniqd` created a list of unique items via keys of a dictionary
- Here is a short way to do this using list comprehension

In [23]:
```
list({ x[2]:1 for x in matchlist})
```

Out[23]:
```
['RSA',
 'CZE',
 'BIH',
 'PAR',
 'SUI',
 'MAR',
 'SCO',
 'TUR',
 'CUW',
 'JPN',
 'ECU',
 'TUN',
 'CPV',
 'EGY',
 'URU',
 'NZL',
 'SEN',
 'NOR',
 'ALG',
 'JOR',
 'COD',
 'CRO',
 'PAN',
 'COL',
 'KOR',
 'QAT',
 'HAI',
 'AUS',
 'CIV',
 'SWE',
 'IRN',
 'KSA',
 'IRQ',
 'AUT',
 'UZB',
 'GHA',
 'MEX',
 'CAN',
 'BRA',
 'USA',
 'GER',
 'NED',
 'BEL',
 'ESP',
 'FRA',
 'ARG',
 'POR',
 'ENG']
```

- Uses the fact that a dictionary `d` when interpreted as a sequence is implcitly `d.keys()`
- `list(d)` looks for a sequence `d`

- Can have multiple generators
- Like nested loops, the left most generator is the outermost loop

In [24]:
```
[ (i,j) for i in range(3) for j in range(4) ]
```

Out[24]:
```
[(0, 0),
 (0, 1),
 (0, 2),
 (0, 3),
 (1, 0),
 (1, 1),
 (1, 2),
 (1, 3),
 (2, 0),
 (2, 1),
 (2, 2),
 (2, 3)]
```

- **Example**: "Small" Pythagorean triples

In [25]:
```
[ (x,y,z) for x in range(1,20) for y in range(1,20) for z in range(1,20) if x*x + y*y
```

Out[25]:
```
[(3, 4, 5),
 (4, 3, 5),
 (5, 12, 13),
 (6, 8, 10),
 (8, 6, 10),
 (8, 15, 17),
 (9, 12, 15),
 (12, 5, 13),
 (12, 9, 15),
 (15, 8, 17)]
```

- Avoid duplicates like `(3,4,5)`, `(4,3,5)` -- ensure that `y > x`
- `y in range(x,20)` -- later generator can use value from an earlier one, like in a nested loop

In [26]:
```
[ (x,y,z) for x in range(1,20) for y in range(x,20) for z in range(x,20) if x*x + y*y
```

Out[26]:
```
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (8, 15, 17), (9, 12, 15)]
```

- Can report the triples in a different order

In [27]:
```
[ (y,x,z) for x in range(1,20) for y in range(x,20) for z in range(x,20) if x*x + y*y
```

Out[27]:
```
[(4, 3, 5), (12, 5, 13), (8, 6, 10), (15, 8, 17), (12, 9, 15)]
```

Scope and global variables

- The scope of a variable refers to the portion of the program where its value is available
- If we refer to a value that is not defined in a function, it is looked up in the global context

In [28]:
```
def f():
    y = x + 22
    print(y)
    return
```

In [29]:
```
# x = 7
f()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[29], line 2
      1 # x = 7
----> 2 f()

Cell In[28], line 2, in f()
      1 def f():
----> 2     y = x + 22
      3     print(y)
      4     return

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

In [30]:
```
x = 7
f()
```

```
29
```

- As soon as we assign a variable a value inside a function, *all* instances of that variable are treated as local to the function
- This decision is *static* based on the program text. In the code below, we cannot be sure that the assignment `x = 33` will execute, but Python still denotes `x` to be local to `f()`
- Though the check is based on the static program text, the error is flagged only when the function executes. The definition of `f()` does not trigger an error though the problem is evident in the text of the function.

In [31]:
```
def f():
    y = x + 22
    print(y)
    if y > 1000:
        x = 33
    return
```

In [32]:
```
x = 7
f()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[32], line 2
      1 x = 7
----> 2 f()

Cell In[31], line 2, in f()
      1 def f():
----> 2     y = x + 22
      3     print(y)
      4     if y > 1000:
      5         x = 33

UnboundLocalError: cannot access local variable 'x' where it is not associated with a
value
```

- This static check applies even if it is impossible for the local assignment to be executed

In [33]:
```
def checky():
    y = x + 2
    return
    if False:
        x = 7
```

In [34]:
```
x = 8
checky()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[34], line 2
      1 x = 8
----> 2 checky()

Cell In[33], line 2, in checky()
      1 def checky():
----> 2     y = x + 2
      3     return
      4     if False:
      5         x = 7

UnboundLocalError: cannot access local variable 'x' where it is not associated with a
value
```

- More examples of using global values within a function without redefining the variable

In [35]:
```
def display_count():
    print(count)
    return
```

In [36]:
```
def display_upto_count():
    for i in range(count):
        print(count+i)
    return
```

- If we call `display_count()` without a global definition for `count` we get an error

In [37]:
```
display_count()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[37], line 1
----> 1 display_count()

Cell In[35], line 2, in display_count()
      1 def display_count():
----> 2     print(count)
      3     return

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

- If `count` is available in the global context, the two functions work as expected

In [38]:
```
count = 7
```

In [39]:
```
display_count()
```

```
7
```

In [40]:
```
display_upto_count()
```

```
7
8
9
10
11
12
13
```

- If we try to update `count` inside the function, both occurrences become local
- The occurrence on the right hand side of the assignment generates an error because its value is now undefined
- Once again, this *static* error is only triggered at run-time when the function executes

In [41]:
```
def increment_local(k):
    count = count+k
    return
```

In [42]:
```
increment_local(2)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[42], line 1
----> 1 increment_local(2)

Cell In[41], line 2, in increment_local(k)
      1 def increment_local(k):
----> 2     count = count+k
      3     return

UnboundLocalError: cannot access local variable 'count' where it is not associated wit
h a value
```

- Reassigning a variable within a function disconnects it from the external variable with the same name

In [43]:
```
def reset_local(k):
    count = k
    return
```

In [44]:
```
reset_local(77)
```

In [45]:
```
count
```

Out[45]: 7

- We can declare a variable to be `global` to override Python's default scope rules
- `global` tells Python to treat the variable inside the function as one from the global context

In [46]:
```
def increment_global(k):
    global count
    count = count+k
    return
```

In [47]:
```
increment_global(8)
```

In [48]:
```
display_count()
```

```
15
```

- The default rule about local scope applies to mutable values as well

In [49]:
```
def concat_local():
    l1 = l1 + l2
    return
```

In [50]:
```
l1 = [1,2,3]
l2 = [4,5,6]
concat_local()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[50], line 3
      1 l1 = [1,2,3]
      2 l2 = [4,5,6]
----> 3 concat_local()

Cell In[49], line 2, in concat_local()
      1 def concat_local():
----> 2     l1 = l1 + l2
      3     return

UnboundLocalError: cannot access local variable 'l1' where it is not associated with a
value
```

In [51]:
```
def concat_global():
    global l1
    l1 = l1 + l2
    return
```

In [52]:
```
l1 = [1,2,3]
l2 = [4,5,6]
concat_global()
```

In [53]:
```
l1, l2
```

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

- We can define a value inside a function and "export" it outside by declaring it `global`

In [54]:
```
del(l1)
del(l2)
```

In [55]:
```
l1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[55], line 1
----> 1 l1

NameError: name 'l1' is not defined
```

In [56]:
```
l2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[56], line 1
----> 1 l2

NameError: name 'l2' is not defined
```

In [57]:
```
def concat_global():
    global l1
    l1 = [1,2,3]
    l1 = l1 + l2
    return
```

In [58]:
```
l2 = [4,5,6]
concat_global()
```

In [59]:
```
l1
```

Out[59]:
```
[1, 2, 3, 4, 5, 6]
```

- The following would work with *dynamic* scoping -- based on execution of program
  - `init()` defines `b` and then calls `seta()`, so with dynamic scoping, `b` is known to `seta()`
- Python uses *static* scoping -- based on text of program -- so this code generates an error
- Most languages use static scoping because dynamic scoping makes it hard to reason about correctness

In [60]:
```
def seta():
    a = b + 5
    print(a)

def init():
    b = 7
    seta()

init()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[60], line 9
      5 def init():
      6     b = 7
      7     seta()
      8
----> 9 init()

Cell In[60], line 7, in init()
      5 def init():
      6     b = 7
----> 7     seta()

Cell In[60], line 2, in seta()
      1 def seta():
----> 2     a = b + 5
      3     print(a)

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