Nested Structures

More Data & Flow Controls

Covered

```
int, float,
str, bool,
List, dict
```

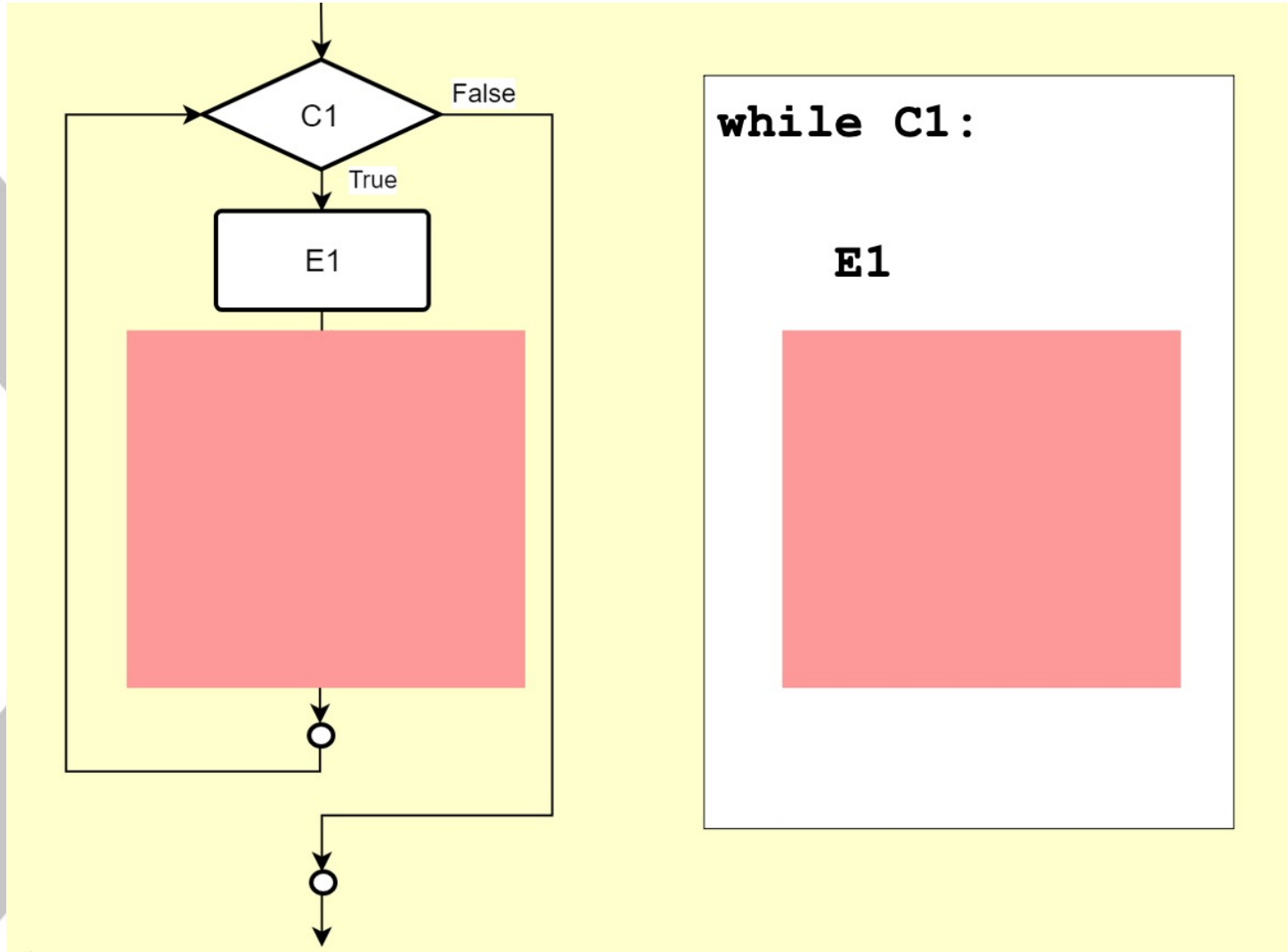
```
if-elef-else
while
for
break
function
```

Will Covered

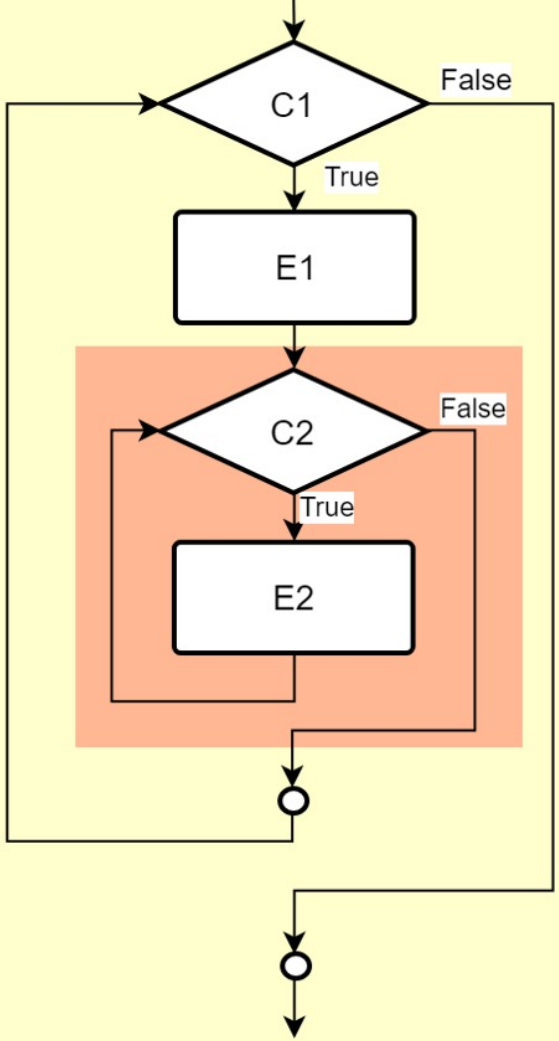
```
tuple, set,
list, dict,
numpy array,
class/object
```

nested loop
comprehension
recursion

Nested loops: nested while(s)



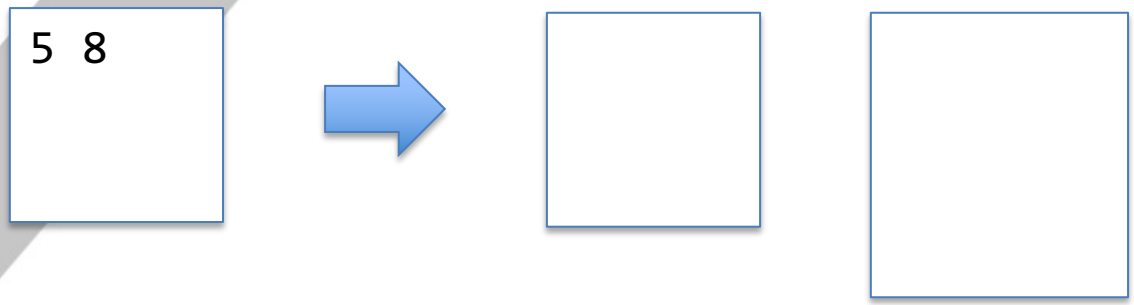
Nested loops: nested while(s)




```
while C1:

    E1

    while C2:

        E2
```

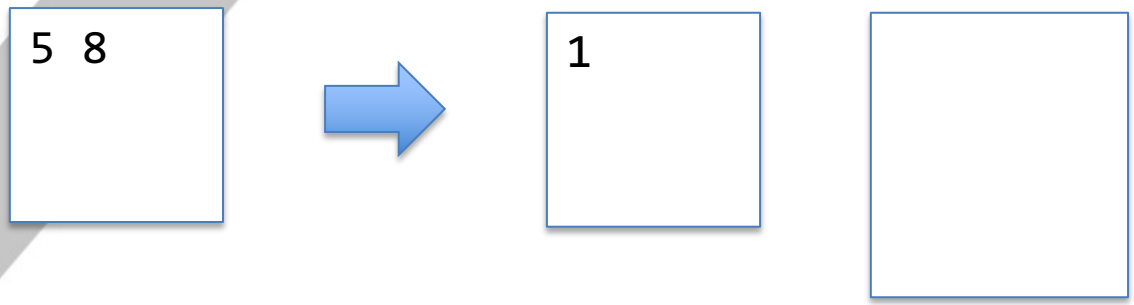
Example: gcd

```
x = input().split()
while x[0] != 'q':



   x = input().split()
```



Example: gcd

```
x = input().split()
while x[0] != 'q':



   x = input().split()
```



Example: gcd

```
x = input().split()
while x[0] != 'q':



   x = input().split()
```

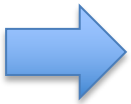
```
5 8
143 65
```

```
1
```

Example: gcd

```
x = input().split()
while x[0] != 'q':



   x = input().split()
```

```
5 8
143 65
```

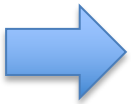


```
1
13
```

Example: gcd

```
x = input().split()
while x[0] != 'q':



   x = input().split()
```

```
5 8
143 65
q
```

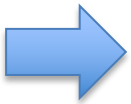


```
1
13
```

Example: gcd

```
x = input().split()
while x[0] != 'q':
    a, b = int(x[0]), int(x[1])
    while b != 0:
        a,b = b, a % b
    print(a)
    x = input().split()
```

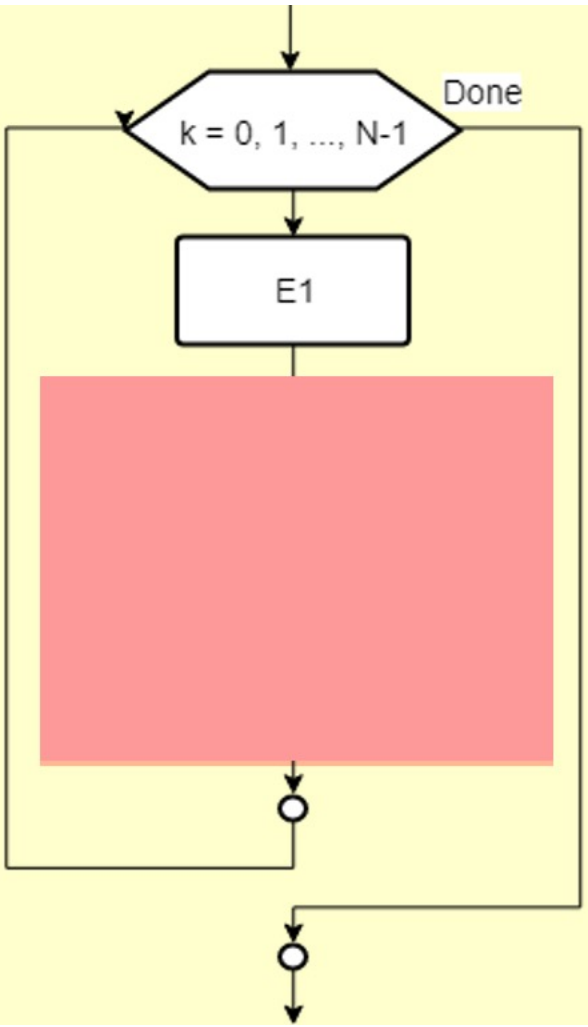
```
5 8
143 65
q
```



```
1
13
```

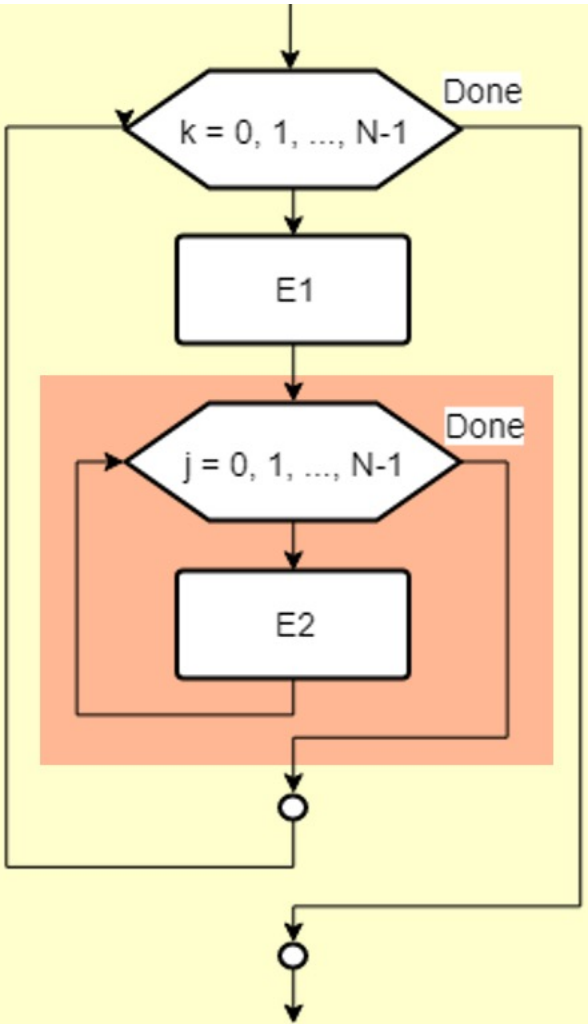
a	b
143	65
65	13
13	0

Nested loops: nested for (s)




```
for k in range(N):

    E1
```

Nested loops: nested for (s)




```
for k in range(N):

    E1

    for j in range(N):

        E2
```

Example

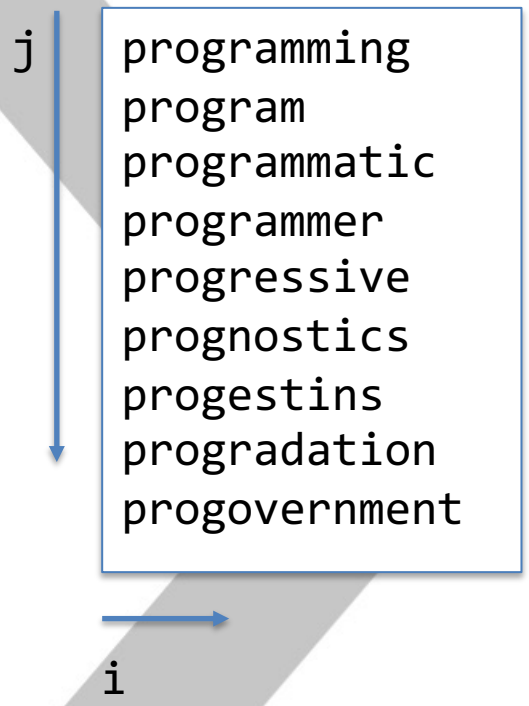
```
for i in range(3):
    for j in range(4):
        print(i, j)
```

```
 i       j
---------
 0       0
         1
         2
         3
 1       0
         1
         2
         3
 2       0
         1
         2
         3
```

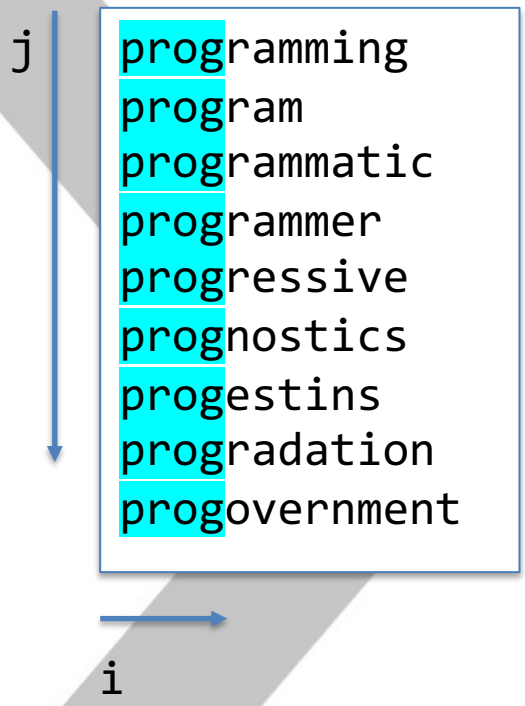
```
for i in range(3):
    for j in range(i, 4):
        print(i, j)
```

```
 i       j
---------
 0       0
         1
         2
         3
 1       1
         2
         3
 2       2
         3
```

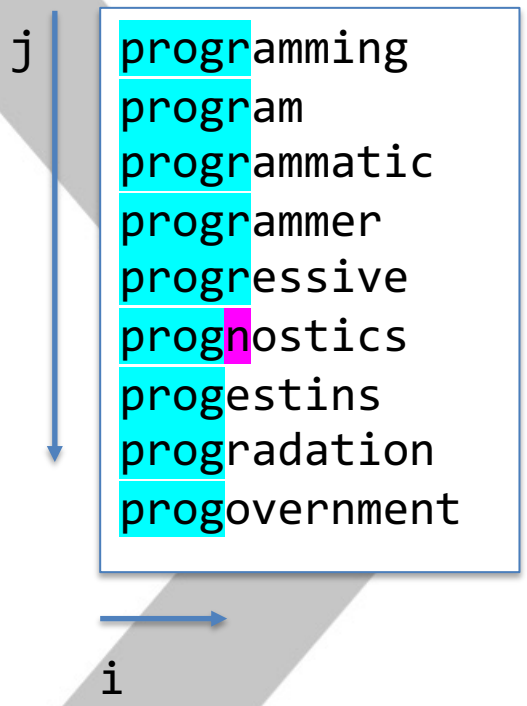
Example: find longest common prefix in string list



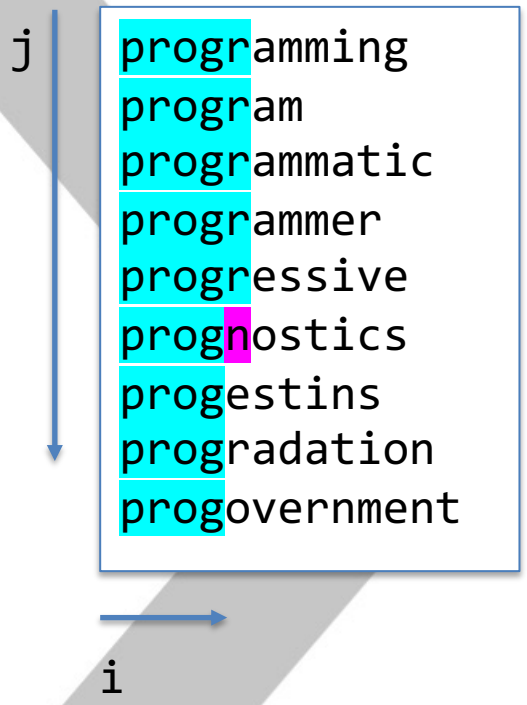
Example: find longest common prefix in string list



Example: find longest common prefix in string list

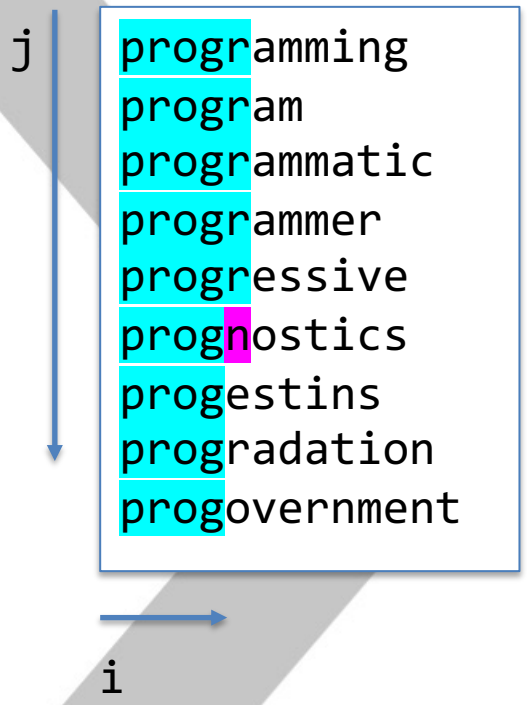


Example: find longest common prefix in string list



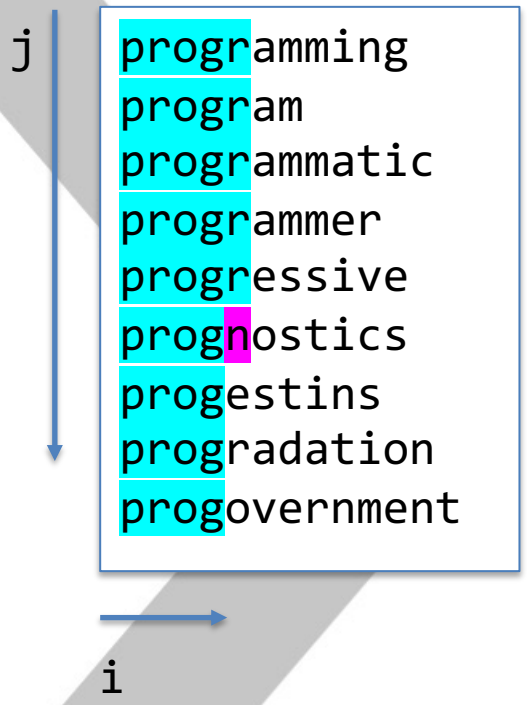

```
def longest_prefix(words):
    for i in range(len(words[0])):
        c = workds[0][i]
        for j in range(1, len(words)):
```

Example: find longest common prefix in string list




```
def longest_prefix(words):
    for i in range(len(words[0])):
        c = workds[0][i]
        for j in range(1, len(words)):
            if i >= len(words[j]) or \
                    c != words[j][i]:
                return words[j][:i]
```

Example: find longest common prefix in string list




```
def longest_prefix(words):
    for i in range(len(words[0])):
        c = workds[0][i]
        for j in range(1, len(words)):
            if i >= len(words[j]) or \
                    c != words[j][i]:
                return words[j][:i]
    return words[0]
```

Example: Function to detect repetition in list


```
def has duplicate ( x ):
```

```
x = [ 11, 34, 22, 34]
```

Example: Function to detect repetition in list


```
def has duplicate ( x ):
```

```
x = [ 11, 34, 22, 34]
```

Example: Function to detect repetition in list


```
def has duplicate ( x ):
```

```
x = [ 11, 34, 22, 34]
```

Example: Function to detect repetition in list


```
def has duplicate ( x ):
```

```
x = [ 11, 34, 22, 34]
```

Example: Function to detect repetition in list


```
def has duplicate ( x ):
```

```
x = [ 11, 34, 22, 34]
```

Example: Function to detect repetition in list


```
def has duplicate ( x ):
```

```
x = [ 11, 34, 22, 34]
```

Example: Function to detect repetition in list

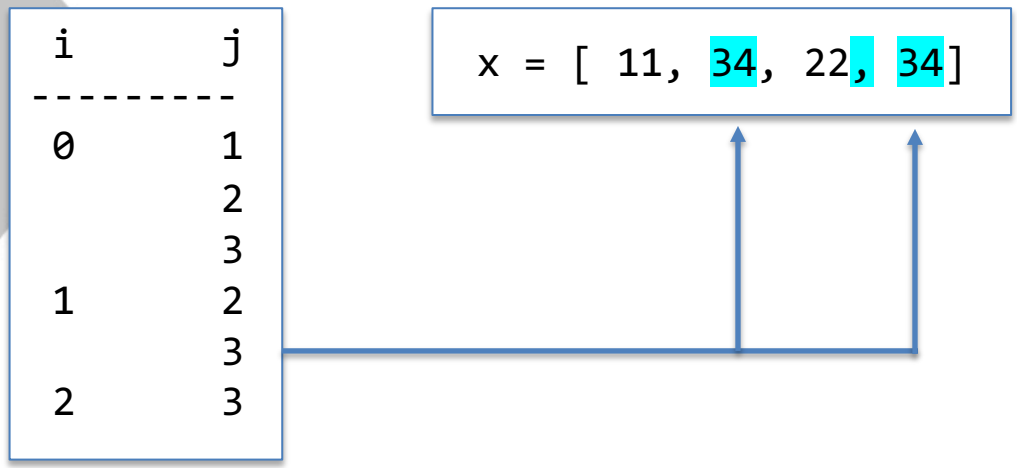

```
def has duplicate ( x ):
```

i	j
0	1
	2
	3
1	2
	3
2	3

```
x = [ 11, 34, 22, 34]
```

Example: Function to detect repetition in list

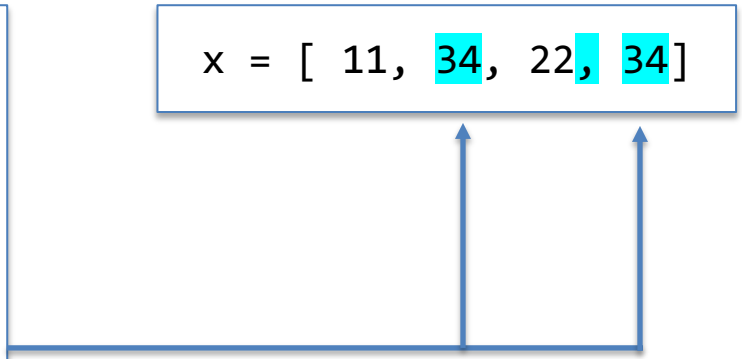

```
def has duplicate ( x ):
```



Example: Function to detect repetition in list


```
def has duplicate ( x ):
    for i in range(len(x) - 1):
        for j in range(i+1, len(x)):
```

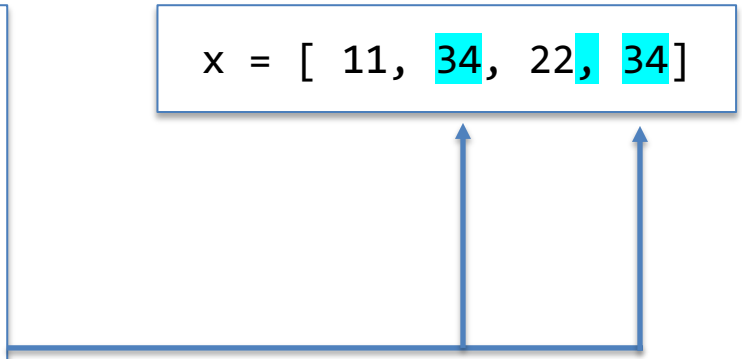
i	j
0	1
	2
	3
1	2
	3
2	3



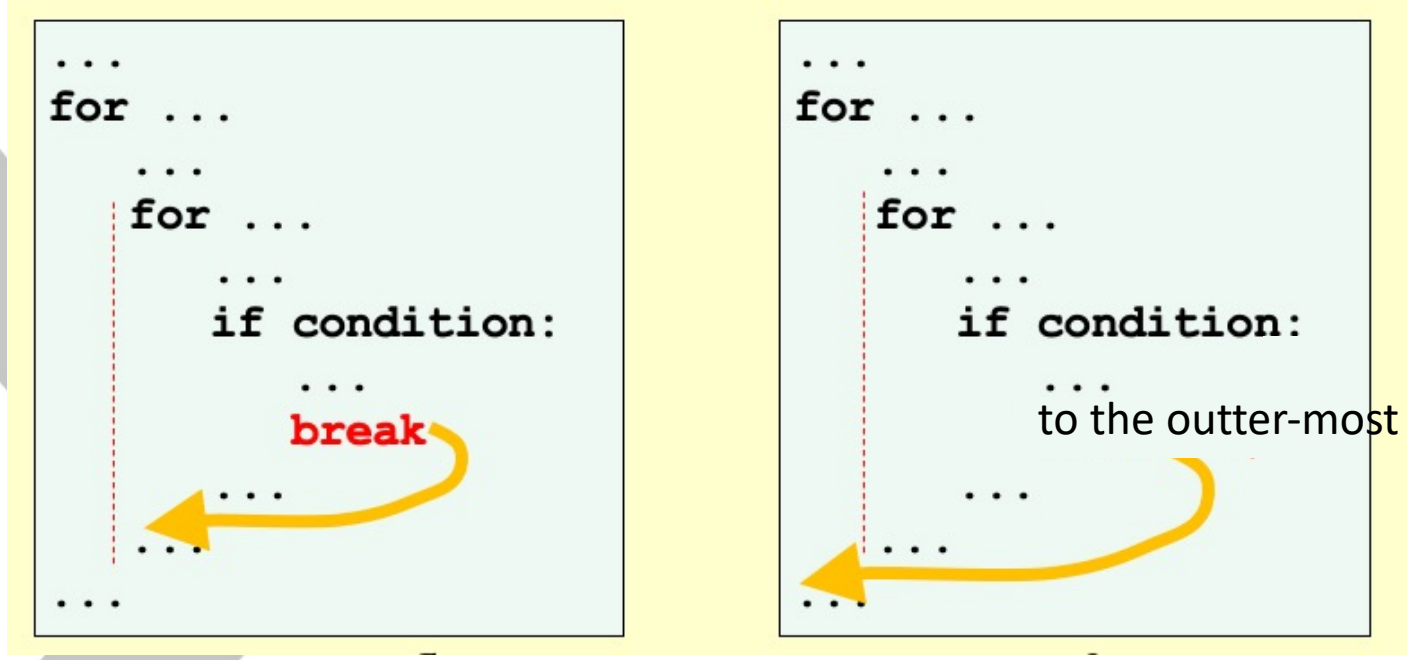
Example: Function to detect repetition in list


```
def has duplicate ( x ):
    for i in range(len(x) - 1):
        for j in range(i+1, len(x)):
            if x[i] == x[j]:
                return True
     return False
```

i	j
0	1
	2
	3
1	2
	3
2	3



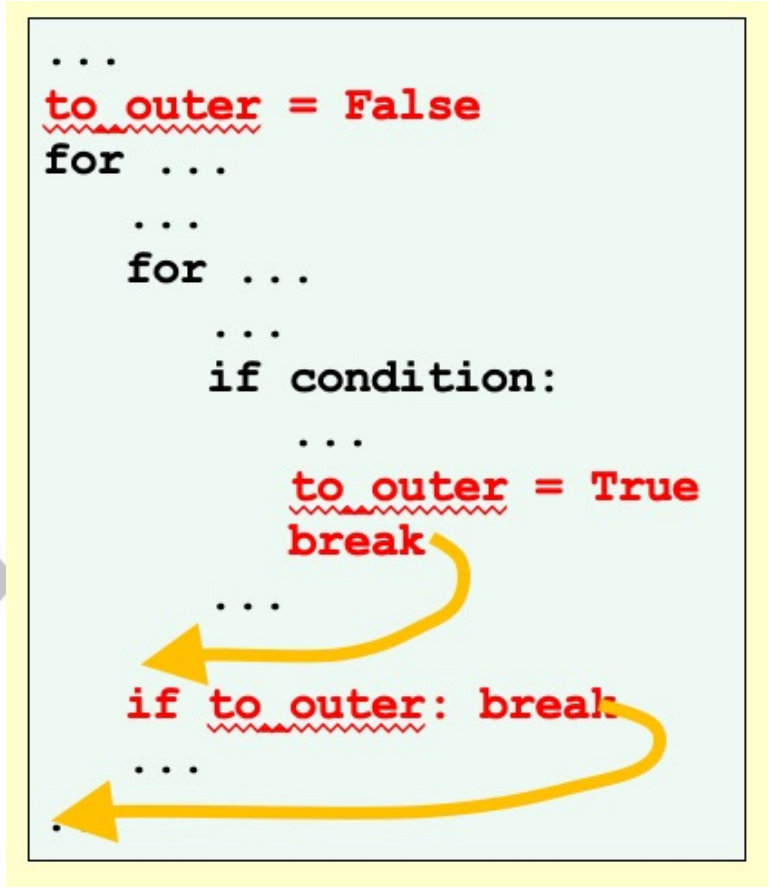
Cannot break out of more-than-one scope



break will jump out of current scope in which it is.

What do you do to jump to the outer-most scope?

Breaking out of multiple nested scopes using auxiliary variable



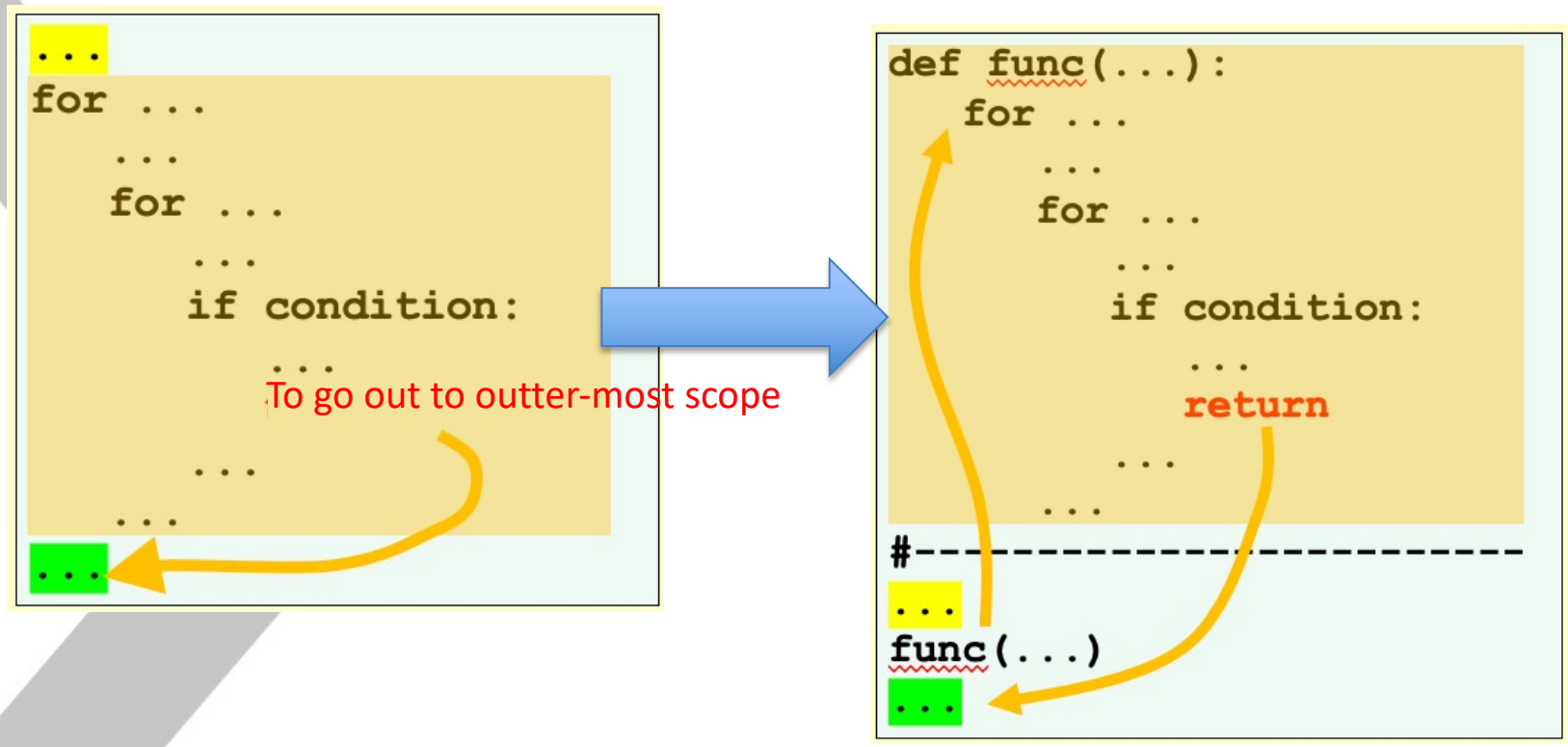
Breaking out of multiple nested scopes using auxiliary variable


```
prefix = words[0]
for i in range(len(words[0])):
    c = words[0][i]
    for j in range(1,len(words)):
        if i >= len(words[j]) or \
            c != words[j][i]:
            prefix = words[j][:i]
            To go out outter-most scope
```

finding longest common prefix

```
prefix = words[0]
found = False
for i in range(len(words[0])):
    c = words[0][i]
    for j in range(1,len(words)):
        if i >= len(words[j]) or \
            c != words[j][i]:
            prefix = words[j][:i]
            found = True
            break
    if found: break
```

Breaking out of multiple nested scopes using function



Breaking out of multiple nested scopes using function


```
prefix = words[0]
for i in range(len(words[0])):
    c = words[0][i]
    for j in range(1,len(words)):
        if i >= len(words[j]) or \
            c != words[j][i]:
            prefix = words[j][:i]
```

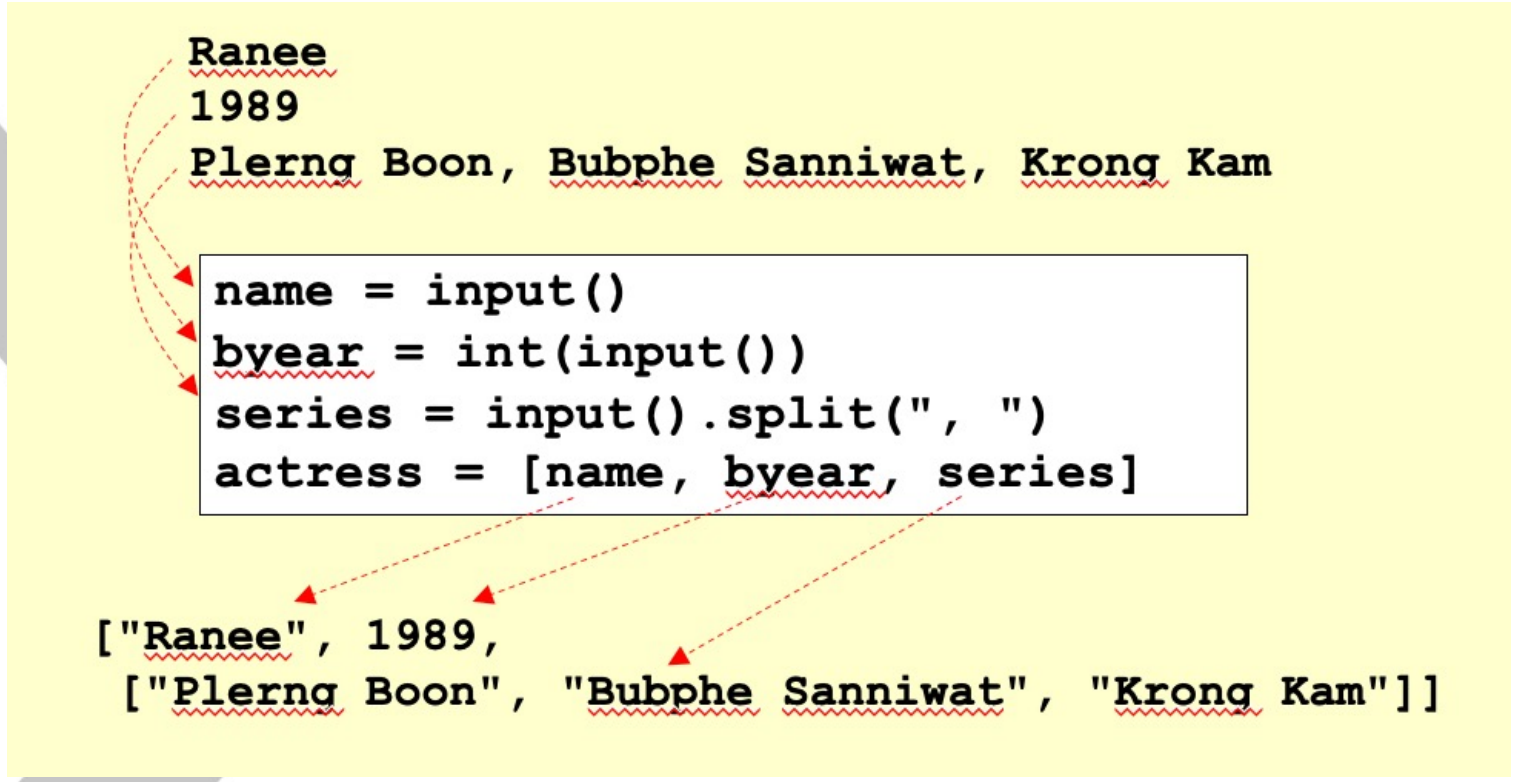
To go out to outter-most scope

```
def longest_prefix(words):
    for i in range(len(words[0])):
        c = words[0][i]
        for j in range(1,len(words)):
            if i >= len(words[j]) or \
                c != words[j][i]:
                return words[j][:i]
    return words[0]

prefix = longest_prefix(words)
```

Nested lists

- To retain data structure, which has some subcomponents as lists
  - `["Renee", 1989,`
    `["Plerng Boon", "Bupphe Sanniwat", "Krong Kam"]]`
- To keep several data structures each with its own subcomponents
  - `[[6131001021, A], [6130020221, B]]`
- To be used as scratch-pad data structure for subsequent processing (e.g., sorting by lengths)
  - `["your", "heart", "is", "on", "my", "list"]`
  - `[[4, "your"], [4, "heart"], [2, "is"],`
    `[2, "on"], [4, "my"], [2, "list"] ]`
- To retain matrix
  - `[[1, 2, 3, 0],`
    `[2, 3, 0, 1],`
    `[4, 1, 2, 2] ]`

Building nested lists



Building nested lists


```
3
6131001021 3.8
6130020221 3.7
6130150721 2.7
```

```
n = int(input())
students = []
for i in range(n):
    student_ID, gpax = input().split()
    gpax = float(gpax)
    students.append( [student_ID, gpax] )
```

```
[["6131001021", 3.8], ["6130020221", 3.7], ["6130150721", 2.7]]
```

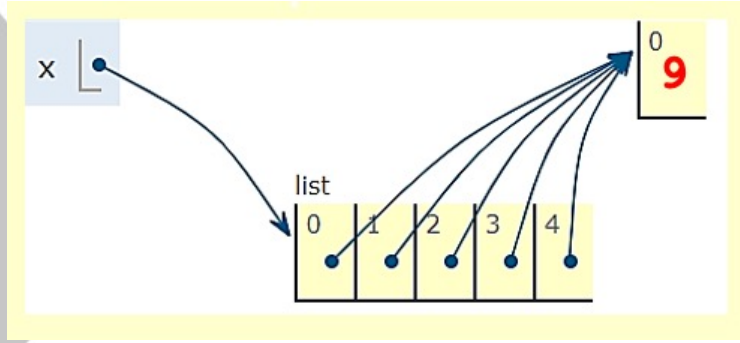
Building nested lists

```
["your", "kiss", "is", "on", "my", "list"]

  [ [4, "your"], [4," kiss"], [2, "is"],
    [2, "on"], [2, "my"], [4, "list"] ]

def sort_by_length( words ):
    x = []
    for w in words:
        x.append( [len(w), w] )
    x.sort()
    for k in range(len(x)):
        words[k] = x[k][1]

  [ [2, "is"], [2, "my"], [2, "on"],
    [4," kiss"], [4, "list"] [4, "your"] ]

["is", "my", "on", "kiss", "list", "your"]
```

Warning

```
x = [0]*5, get [0, 0, 0, 0]
```

```
x = [[0]]*5, get [[0], [0], [0], [0], [0]]
x[0][0] = 9, get [[9], [9], [9], [9], [9]]
```




```
x = []
for i in rage(5):
   x.append([0])
```

for this one, each item is separate list

Nested list as matrix

```
3                       [ [1.0, 2.0, 3.0, 0.0],
1 2 3 0                   [2.0, 3.0, 0.0, 1.0],
2 3 0 1                   [4.0, 1.0, 2.0, 2.0] ]
4 1 2 2
```

```
def read_matrix():
    m = []
    nrows = int(input())
    for k in range(nrows):
        x = input().split()
        r = []
        for e in x:
            r.append( float(e) )
        m.append(r)
    return m
```

print_matrix(M)

```
def print_matrix( M ):
    if len(M) == 1:
        print(M)
    else:
        print("[" + str(M[0]))
        for i in range(1,len(M)-1):
            print(" " + str(M[i]))
        print(" " + str(M[-1]) + "]")
```

```
M = [[1,2,3,4],[2,2,1,3],[2,6,7,7]]
print_matrix(M)
```

```
[[1, 2, 3, 4]
 [2, 2, 1, 3]
 [2, 6, 7, 7]]
```

add_matrix(A, B)

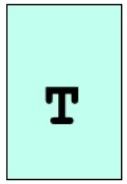
```
def add_matrix(A, B):
    C = []
    nrows = len(A)
    ncols = len(A[0])
    for i in range(nrows):
        C.append( [0.0]*ncols )
        for j in range(ncols):
            C[i][j] = A[i][j] + B[i][j]
    return C
```

```
A = read_matrix()
B = read_matrix()
C = add_matrix(A, B)
print_matrix(C)
```

identity(r) & transpose(M)


```
def identity( n ):
    I = []
    for r in range( n ):
        I.append( [0]*n )
        I[r][r] = 1
    return I
```

```
def transpose( A ):
    T = []
    nrows = len(A)
    ncols = len(A[0])
    for c in range(ncols):
        T.append([0]*nrows)
        for r in range(nrows):
            T[c][r] = A[r][c]
    return T
```



List comprehension

For x be a link of int

```
t = []
for e in x:
   t.append(2*e)
```

```
# change every value in x to other value

t = [ 2*e  for e in x ]
```

```
t = []
for e in x:
   if e >= 0:
      t.append(e)
```

```
# select some values in list x

t = [ e for e in x if e >= 0 ]
```

```
t = []
for e in x:
   if e >= 0:
      t.append(2*e)
```

```
# select some values in list x to modify

t = [2*e for e in x if e >= 0 ]
```

Example: reading line of numbers to list


```
x = input().split()
d = []
for e in x:
    d.append( int(e) )
...
```

```
d = []
for e in input().split():
    d.append( int(e) )
...
```

Using list comprehension

```
d = [int(e) for e in input().split()]
...
```

```
d = [float(e) for e in input().split()]
...
```

Example: sorting strings from their lengths


```
def sorted_by_length(s):
    t = []
    for e in s:
        t.append( [len(e), e] )
    t.sort()
    r = []
    for n,e in t:
        r.append( e )
    return r
```

Using list comprehension

```
def sorted_by_length(s):
    t = [[len(e),e] for e in s]
    t.sort()
    return [e for n,e in t]
```

List comprehension is faster

```
import timeit

def for_loop(n):
    t = []
    for i in range(n):
        t.append(n)
    return t

def comprehension(n):
    return [n for i in range(n)]

def time(func):
    print(timeit.timeit(func+"(1000000)",
          globals=globals(), number=100))

time("for_loop")       # 9.2609192
time("comprehension") # 4.7720880999999995
```