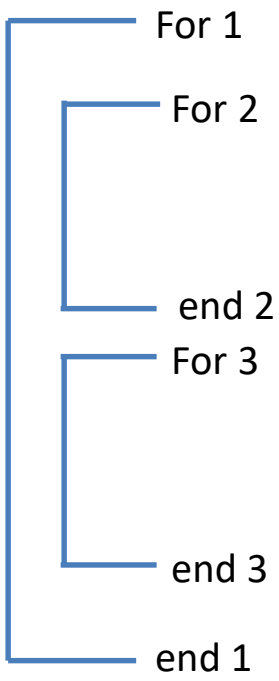
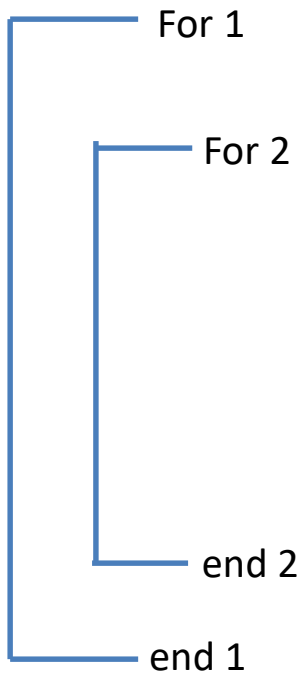
Nested loops

Nested Loop Pattern

The indented code block of a For loop can contain any sequence of Python statements, including another For loop. However, the second loop must be completely contained inside the first loop and have a different control variable. Such a sequence is called a **nested For Loop**.





Figurative pattern: square

```
def square1(n):
    'draw n x n solid square'
    for i in range(n):
        for j in range(n):
            print('*', end='  ')
        print()
```

Add an IF statement in the inner (nested) loop

Figurative pattern: triangle

```
def  triangle(n):
   'draw right angle triangle'
   for i in range(n):
      for j in range(i+1):
          print('*', end='  ')
       print()
```

List of lists

[[‘CSC’, ‘IT’, ‘IS’], [101, 130, 120, 399]]

```
CSC 101
CSC 130
CSC 120
CSC 399
IT 101
IT 130
IT 120
IT 399
IS 101
IS 130
IS 120
IS 399
```

```
def comb(lst1, lst2):
    for label in lst1:
        for number in lst2:
            formatStr = '{} {}'
            print(formatStr.format(label,number))
    return
```

```
comb(['CSC', 'IT', 'IS'] , [101, 130, 120, 399])
```

List of lists

[[‘CSC’, ‘IT’, ‘IS’], [101, 130, 120, 399]]

```
def comb2(lst1, lst2):
    outlst = []
    for label in lst1:
        for number in lst2:
            formatStr = '{} {}'
            outlst.append(formatStr.format(label, number))
    return outlst
```

make 1 list

```
xlist =comb2(['CSC', 'IT', 'IS'] , [101, 130, 120, 399])
print(xlist)
```

['CSC 101', 'CSC 130', 'CSC 120', 'CSC 399', 'IT 101', 'IT 130', 'IT 120', 'IT 399', 'IS 101', 'IS 130', 'IS 120', 'IS 399']

Two-dimensional lists

The list `[3, 5, 7, 9]` can be viewed as a 1-D table

`[3, 5, 7, 9]` =

3	5	7	9

How to represent a 2-D table?

```
[ [3, 5, 7, 9]
  [0, 2, 1, 6]
  [3, 8, 3, 1] ]
```

	0	1	2	3
0	3	5	7	9
1	0	2	1	6
2	3	8	3	1

A 2-D table is just a list of rows (i.e., 1-D tables)

```
>>> lst = [[3,5,7,9],
           [0,2,1,6],
           [3,8,3,1]]
>>> lst
[[3, 5, 7, 9],
[0, 2, 1, 6],
[3, 8, 3, 1]]
>>> lst[0]
[3, 5, 7, 9]
>>> lst[1]
[0, 2, 1, 6]
>>> lst[2]
[3, 8, 3, 1]
>>> lst[0][0]
3
>>> lst[1][2]
1
>>> lst[2][0]
3
>>>
```

Multi-dimensional lists

lst = [[12, 19, -2], [-3, 23, 4], [2, 2, 2, 1]]

Each sublist is a row in the table

```
def suml(lst):
    'sum all elements in sublists of lst '
    s = 0
    for row in lst:
        for elem in row:
            s += elem
    return s
```

Returns 60

Trace variables

row	s = s + elem

What is:
lst[0][0]
lst[1][2]

List Loop Iteration

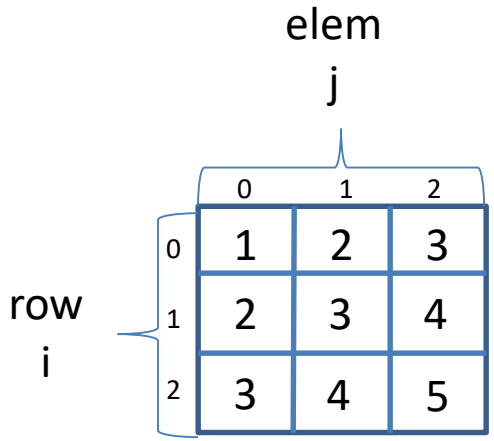
By value

By index

lst = [[1,2,3], [2,3,4], [3,4,5]]

```
for row in lst:
    x= 0
    for elem in row:
        x += elem
    print (x)
```

```
for i in  range(0, len(lst)):
    x = 0
    for j in range(0, len(lst)):
        x += lst[i][j]
    print (x)
```



Creating 2D lists

To create list 'ages' in your program, you first need to write
ages = [ ], i.e. create an empty list.

To create a 2D list in your program, you first need to the following:
For example, to create a 5 x 2 matrix:

```
t=[]
for x in range(5):     # number of sublists (rows)
   t.append([])        # creates the empty sublist
   for y in range(2): # number of items in each sublist
      t[x].append(0)
```

[[0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]

Note: x is correct.; this code is appending 0's to the x row

To change specific items in the 2D list, we can now use assignment statements, for examples:

```
t[0][1] = 30
t[4][0] = 10
```

[[0, 30], [0, 0], [0, 0], [0, 0], [10, 0]]

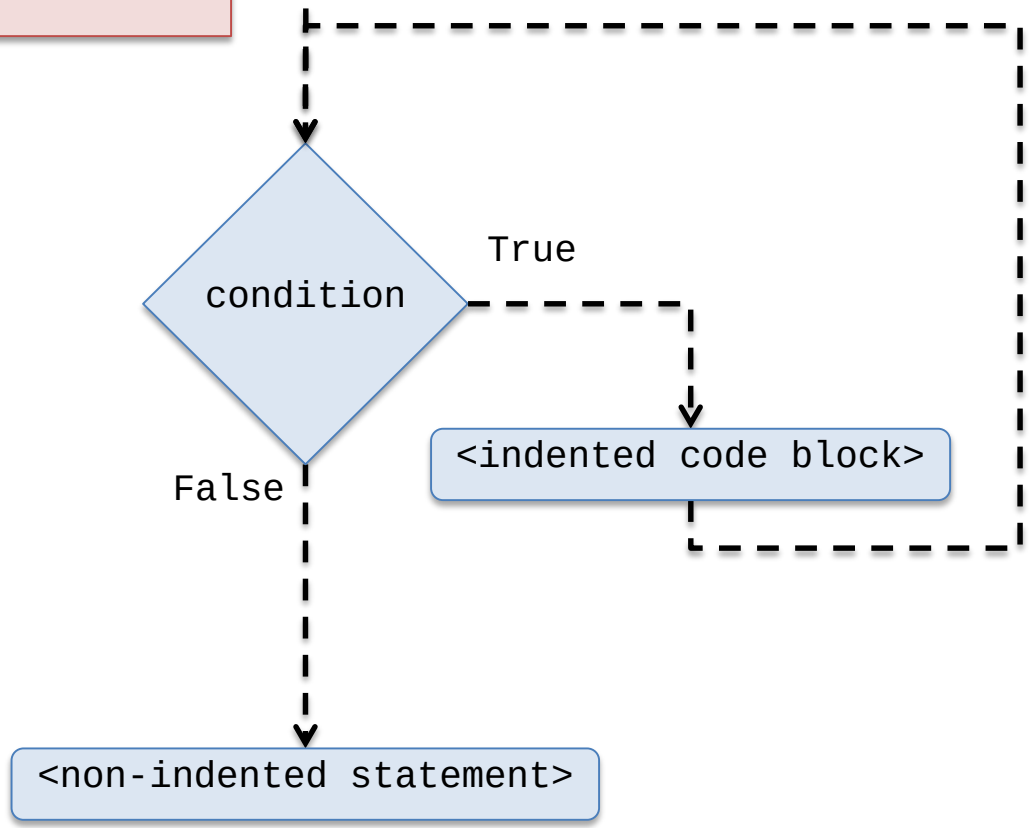
While loop

Looping on the unknown

- When you need to be able to repeat a block a code for a number of times that is not known at a start, you need a different loop structure
- WHILE LOOP

while loop

```
while <condition>:
    <indented code block>
<non-indented statement>
```



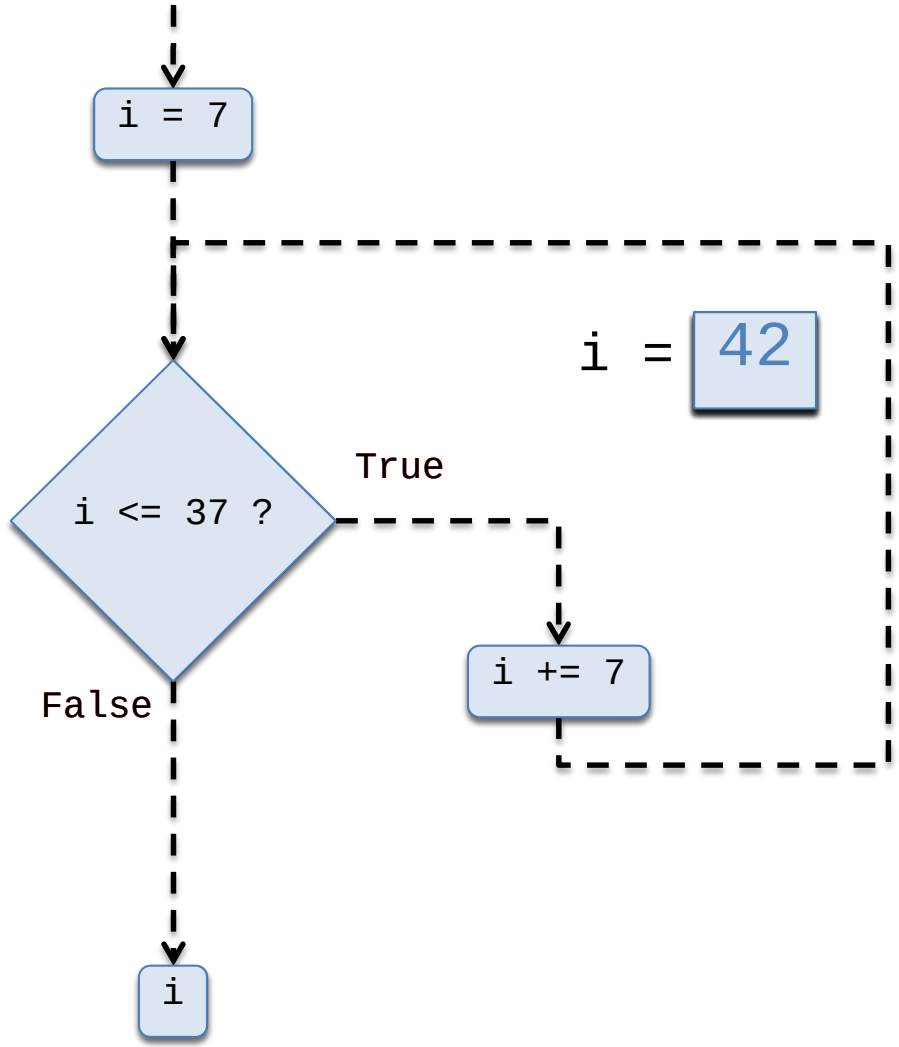
while loop

Example: compute the smallest multiple of 7 greater than 37.

Idea: generate multiples of 7 until we get a number greater than 37

```
>>> i = 7
>>> while i <= 37:
        i += 7

>>> i
42
```



Trace variables

i	lst[i]

Sequential Loop Pattern

Return first positive number in lst

```
def firstpos(lst):
    'return first positive number in lst‘

    i = 0

    while (lst[i] <= 0):
        i += 1
    return lst[i]
```

```
>>> firstpos([-1, -2, -3])
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    firstpos([-1, -2, -3])
  File "C:/Users/dkalayta/Documents/CSC 401 -
ile_examples.py", line 7, in firstpos
    while (lst[i] <= 0):
IndexError: list index out of range
```

Trace variables

i	lst[i]

```
>>> firstpos([-9, -8, -7, 6, 5, 4, -3, 2, 1])
6
```

Trace variables

i	len(lst) and lst[i]

Sequential Loop Pattern

Return first positive number in lst

```
def firstpos(lst):
    'return first positive number in lst‘

    i = 0

    while (i < len(lst) and lst[i] <= 0):
        i += 1
    return lst[i]
```

```
>>> firstpos([-1, -2, -3])
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <modu
    firstpos([-1, -2, -3])
  File "C:/Users/dkalayta/Documents/CS
ile_examples.py", line 20, in firstpos
    return lst[i]
IndexError: list index out of range
```

Trace variables

i	len(lst) and lst[i]

```
>>> firstpos([-9, -8, -7, 6, 5, 4, -3, 2, 1])
6
```

Sequential Loop Pattern

firstpos = [-1, -2, -3]

Return first positive number in lst

```
def firstpos(lst):
    'return first positive number in lst‘

    i = 0

    while (i < len(lst) and lst[i] <= 0):
        i += 1
     if i < len(lst):
        return lst[i]
     else:
        return
```

Trace variables

i	len(lst) and lst[i]	i < len(lst)

```
>>> firstpos([-9, -8, -7, 6, 5, 4, -3, 2, 1])
6
```

Sequential Loop Pattern

Find position or first upper case word in a list

```
def firstup(lst):
    'find position of first upper case word in lst'

    i = 0

    while (i < len(lst) and not lst[i][0].isupper()):
        i += 1
    if i < len(lst):
        return i
    else:
        return -1
```

Trace variables

i	i< len(lst)	not lst[i][0].isupper()

```
>>> firstup(['hello', 'good', 'Bye', 'John'])
2
>>> firstup(['hello', 'good'])
-1
```

Loop-and-a-half pattern

```
def average():
    'calculate average of numbers entered by user'

    s = 0
    ct = 0

    while (True):
        x = input('Number (or return): ')
        if x == '':
            break  # last loop end here
        s += eval(x)
        ct += 1

    return (s/ct)
```

```
Number (or return): 10
Number (or return): 15
Number (or return): 25
Number (or return):
16.666666666666668
```

Interactive Loop

Cutting the last loop iteration “in half”

The empty string is a “flag” that indicates the end of the input

The `break` statement

The `break` statement:

- is used inside the body of a loop
- when executed, it interrupts the current iteration of the loop
- execution continues with the statement that follows the loop body.

```
def cities2():
    lst = []

    while True:
        city = input('Enter city: ')

        if city == '':
            return lst

        lst.append(city)
```

```
def cities2():
    lst = []

    while True:
        city = input('Enter city: ')

        if city == '':
            break

        lst.append(city)

    return lst
```

Infinite loop pattern

An infinite loop provides a continuous service

```
>>> hello2()
What is your name? Sam
Hello Sam
What is your name? Tim
Hello Tim
What is your name? Alex
Hello Alex
What is your name?
```

A greeting service

The server could instead be a time server, or a web server, or a mail server, or…

```
def hello2():
    '''a greeting service; it repeatedly requests the name
       of the user and then greets the user'''

    while True:
        name = input('What is your name? ')
        print('Hello {}'.format(name))
```

Interactive Loop

w	long_word	len(w)	len(long_word)

```
def long_words():
    'return longest word entered by user'

    w = input('Word: ')
    long_word = ''
    while(len(w)> 0):
        if len(w) > len(long_word):
            long_word = w
        w = input('Word: ')
    return long_word
```

```
>>> long_words()
Word: Hello
Word: two
Word: goodbye
Word: three
Word: four
Word: five
Word: six
Word: seven
Word: eight
Word: nine
Word:
'goodbye'
```

Interactive Loop

w	long_word	len(w)	len(long_word)

```
def short_words():
    'return shortest word entered by user'

    w = input('Word: ')
    short_word = ''
    while(len(w)> 0):
        if len(w) < len(short_word):
            short_word = w
        w = input('Word: ')
    return short_word
```

```
>>> short_words()
Word: Hello
Word: two
Word: goodbye
Word: three
Word: four
Word: five
Word: six
Word: seven
Word: eight
Word: nine
Word:
''
```

Example

- Write a function that does the following:
  - Stores a fixed number between 0 and 10 in a variable
  - As long as the user says she wants to play:
    - Ask the user to guess a number between 0 and 10
    - Tell the user if she guessed the number you had stored or not
- Write a version of the program that does NOT use while-true and break and one that does