Range Construction

- The **range** function is used to create a sequence of evenly spaced **integers**.
- range(n) creates a sequence of n integers from 0 to n-1
- range(10) creates a sequence of the 10 integers from 0 to 9

```
In [1]: print(list(range(10)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- range(start, stop, step) creates a sequence whose first value is start, second value is start+step, third value is start+2×step, etc., up to but not including stop

```
In [2]: print(list(range(1,20,2)))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

```
In [1]: print(list(range(2,8,3)))
[2, 5]
```

- range(start, stop) is the same as range(start, stop, 1)

```
In [2]: print(list(range(5,10)))
[5, 6, 7, 8, 9]
```

- A range can be used in a for loop and is of the form:

```
for var in range(start, stop, step):
```

where *var* is some variable name and *start* and *step* are optional.

- Modify the Celsius to Fahrenheit program to use a range (toFahrenheitRange.py)

```
from time import ctime

print('\n' + '-'*80)
COLDEST = -40
HOTEST = 40
DELTA_T = 5

print("\nCelsius\tFahrenheit")  # display a heading

for celsius in range(COLDEST, HOTEST+1,DELTA_T):
    fahrenheit = (9./5)*celsius + 32
    print("%7g\t%10g" % (celsius, fahrenheit))

print("""
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

- The output from the program is:

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

Celsius Fahrenheit
    -40        -40
    -35        -31
    -30        -22
    -25        -13
    -20         -4
    -15          5
    -10         14
     -5         23
      0         32
      5         41
     10         50
     15         59
     20         68
     25         77
     30         86
     35         95
     40        104

Programmed by Stew Dent.
Date: Sat May 26 09:09:14 2018.
End of processing.
```

Example

```
>>> # One parameter
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> # Two parameters
>>> for i in range(3, 6):
...     print(i)
...
3
4
5
>>> # Three parameters
>>> for i in range(4, 10, 2):
...     print(i)
...
4
6
8
>>> # Going backwards
>>> for i in range(0, -10, -2):
...     print(i)
...
0
-2
-4
-6
-8
```

Example

- Write a program that can count from one to five like : *one,two,three….*

```
>>> my_list = ['one', 'two', 'three', 'four', 'five']
>>> my_list_len = len(my_list)
>>> for i in range(0, my_list_len):
...     print(my_list[i])
...
one
two
three
four
five
```

Example

- Write a code that continue receiving numbers unless the user enter an even number

Example

```
#!/usr/bin/python

var = 100
if var == 200:
   print "1 - Got a true expression value"
   print var
elif var == 150:
   print "2 - Got a true expression value"
   print var
elif var == 100:
   print "3 - Got a true expression value"
   print var
else:
   print "4 - Got a false expression value"
   print var

print "Good bye!"
```

Example

- Write a code that continue receiving numbers unless the user enter an even number. Otherwise it prints “Enter another number”

- Suppose rather than temperatures in the sequence -40, -35, …, 40 we want temperatures in the sequence -20, -17.5, - 15, -12.5, …, 20
- The range function can only create a sequence of integers, which can then be converted into the desired numbers in the sequence -20, -17.5, - 15, -12.5, … 20
- Convert **0** to -20, **1** to -17.5, **2**, to -15, … **16** to 20.
- -20 + **0** * 2.5 = -20
- -20 + **1** * 2.5 = -17.5
- -20 + **2** * 2.5 = -15
- -20 + **16** * 2.5 = 20
- The required range of integer values is 0,1, …, 16 or (20 - (-20)) / 2.5, where 2.5 is the difference in the values in the sequence. (toFahrenheitRange1.py)

```
from time import ctime

print('\n' + '-' * 80)
COLDEST = -20
HOTEST = 20
DELTA_T = 2.5

print("\n%7s %12s" % ('Celsius', 'Fahrenheit'))

for value in range(int((HOTEST-COLDEST)/DELTA_T)+1):
    celsius = COLDEST + value * DELTA_T
    fahrenheit = (9./5)*celsius + 32
    print("%7.1f %12.1f" % (celsius, fahrenheit))

print("""
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

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

Celsius Fahrenheit
  -20.0       -4.0
  -17.5        0.5
  -15.0        5.0
  -12.5        9.5
  -10.0       14.0
   -7.5       18.5
   -5.0       23.0
   -2.5       27.5
    0.0       32.0
    2.5       36.5
    5.0       41.0
    7.5       45.5
   10.0       50.0
   12.5       54.5
   15.0       59.0
   17.5       63.5
   20.0       68.0

Programmed by Stew Dent.
Date: Sat May 26 09:12:38 2018.
End of processing.
```

- Suppose we wish to process corresponding elements of two lists of the same length (often referred to as parallel lists).
- For example one list might contain surnames and another list the corresponding first names, and we want to print out the full names one per line.
- Need to have an index for each element of the lists.
- The index must range from 0 to one less than the length of the lists.
- Example: listIndex.py

```
from time import ctime

print('\n' + '-' * 80)

surnames = 'Jones', 'Smith', 'Taylor', 'Bond',
'Baker'
firstNames = 'Davey', 'Joan', 'Abbey', 'James',
'Bobby'

for i in range(len(surnames)):
    print(firstNames[i], surnames[i])
# for

print("""
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

- The output from the program is:

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

Davey Jones
Joan Smith
Abbey Taylor
James Bond
Bobby Baker

Programmed by Stew Dent.
Date: Sat May 26 09:15:55 2018.
End of processing.
```

- Example - initialize the elements in a list: initList.py

```
from time import ctime

print('\n'+ '-' * 80)

N = 17
numbers = []  # the empty list

for i in range(N):
    numbers.append(-20 + i * 2.5)
# for

print('index  number')
for i, number in enumerate(numbers):
    print('%5d  %6.1f' % (i, number))
# for

print("""
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

- The output from this program is:

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

index  number
    0   -20.0
    1   -17.5
    2   -15.0
    3   -12.5
    4   -10.0
    5    -7.5
    6    -5.0
    7    -2.5
    8     0.0
    9     2.5
   10     5.0
   11     7.5
   12    10.0
   13    12.5
   14    15.0
   15    17.5
   16    20.0

Programmed by Stew Dent.
Date: Sat May 26 09:17:20 2018.
End of processing.
```

- Consider the statements: (e.g. emunerate.py)

```
print('index  number')
for i, number in enumerate(numbers):
    print('%5d  %6.1f' % (i, number))
```

- The **enumerate** function gives the index and corresponding value when looping through a sequence.
- The **enumerate** function can only be used in loops.

```
In [1]: enumerate(numbers)
<enumerate object at 0x1723d78>
```

- If `numbers` contains [50, 60, 70, 80] then `enumerate(numbers)` is [(0, 50), (1, 60), (2, 70), (3, 80)]
- First time through the loop i=0, number=50
- Second time through the loop i=1, number=60
- Third time through the loop i=2, number=70
- Fourth time through the loop i=3, number=80

```
In [1]: numbers = [0, 10, 20, 30, 40]
In [2]: print(numbers)
[0, 10, 20, 30, 40]
In [3]: for number in numbers:
            number += 5
In [4]: print(numbers)
[0, 10, 20, 30, 40]
```

- Notice that the statement

  `number += 5`

did not change the content of *numbers*. The variable *number* is a copy of a value in *numbers* and not an element of *numbers*. Therefore changing the value of *number* does not change the value of any of the elements in *numbers*.

- To change the values of the elements of *numbers* do the following:

```
In [1]: print(numbers)
[0, 10, 20, 30, 40]
In [2]: for i in range(len(numbers)):
            numbers[i] += 5
In [3]: print(numbers)
[5, 15, 25, 35, 45]
```

- Or do the following:

```
In [4]: print(numbers)
[0, 10, 20, 30, 40]
In [5]: for i, number in enumerate(numbers):
            numbers[i] = number + 5
In [6]: print(numbers)
[5, 15, 25, 35, 45]
```

- **List comprehension** is a compact way to create new lists from ranges or other lists.

```
In [1]: oddNums = [2*num+1 for num in
range(10)]
In [2]: print(oddNums)
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
In [3]: evenNums = [oddNum+1 for oddNum in oddNums]
In [4]: print(evenNums)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
In [5]: sumNums = [oddNum + evenNum for oddNum,
          evenNum in zip(oddNums, evenNums)]
In [6]: print(sumNums)
[3, 7, 11, 15, 19, 23, 27, 31, 35, 39]
```

- The ***zip*** function turns *n* lists into one list of tuples where each tuple contains *n* elements (known as an *n-tuple*).

```
In [1]: al = [1,2,3]
In [2]: bl = [9,7,6]
In [3]: print(list(zip(al, bl)))
[(1, 9), (2, 7), (3, 6)]
In [4]: for a, b in zip(al, bl):
            print(a, b)
1 9
2 7
3 6
```

Demo listIndex1.py

Nested Lists

- An element in a list can be of any type including another list.

```
In [1]: la = [1,2,3,4]
In [2]: lb = ['a', 'b', 'c']
In [3]: lc = [la, lb, [.5, -1.7], [True, False]]
In [4]: print(lc)
[[1, 2, 3, 4], ['a', 'b', 'c'], [0.5, -1.7], [True,
False]]
In [5]: for index, element in enumerate(lc):
            print('lc[%d] = %r' %(index, element))
lc[0] = [1, 2, 3, 4]
lc[1] = ['a', 'b', 'c']
lc[2] = [0.5, -1.7]
lc[3] = [True, False]
```

- Notice the use of %r

```
In [1]: for index1, sublist in enumerate(lc):
            for index2, element in
enumerate(sublist):
                print('lc[%d][%d] = %r' % (index1,
                                           index2, element))
```

```
lc[0][0] = 1
lc[0][1] = 2
lc[0][2] = 3
lc[0][3] = 4
lc[1][0] = 'a'
lc[1][1] = 'b'
lc[1][2] = 'c'
lc[2][0] = 0.5
lc[2][1] = -1.7
lc[3][0] = True
lc[3][1] = False
```

```
In [1]: for index1, sublist in enumerate(lc):
            for index2, element in
enumerate(sublist):
                print('lc[%d][%d] = %r' % (index1,
                                           index2, element))
```

- As `lc` is a list whose elements are also lists then `sublist` is a list.
- As `sublist` is a list it is a valid argument to the *enumerate* function.
- As lc[0] = `[1, 2, 3, 4]`

  lc[0][0] = 1

  lc[0][1] = 2

  lc[0][2] = 3

  lc[0][3] = 4

Tables

- A table is a list of lists where each of the lists contains the same number of elements and all of the elements are of the same type.
- A table is usually thought of as having rows and columns, where each row is a list.

```
In [1]: table = [[1, 12, 7, 5],   # 1st row (0)
                 [-5, 6, -1, 9],  # 2nd row (1)
                 [10, 2,  8, 4]]  # 3rd row (2)
In [1]: print(table[0][0])
1
In [1]: print(table[1][2])
-1
In [1]: print(table[2][3])
4
```

- This table contains three lists (rows) each with 4 elements, so the table has 3 rows and 4 columns.

- Create two temperature lists of the same length

```
In [1]: tempC = [-20 + temp * 5 for temp in range(9)]
In [2]: tempF = [9/5. * temp + 32 for temp in tempC]
In [3]: print(len(tempC), tempC)
9 [-20, -15, -10, -5, 0, 5, 10, 15, 20]
In [4]: print(len(tempF), tempF)
9 [-4.0, 5.0, 14.0, 23.0, 32.0, 41.0, 50.0, 59.0, 68.0]
```

- Create a table from the two lists

```
In [5]: temps = [tempC, tempF]
In [6]: print(temps)
[[-20, -15, -10, -5, 0, 5, 10, 15, 20], [-4.0,
5.0, 14.0, 23.0, 32.0, 41.0, 50.0, 59.0, 68.0]]
```

- The table contains two lists (rows) each with 9 elements, so the table has 2 rows and 9 columns.

- If we display the rows of the table we get

```
In [1]: for row in temps:
            print(row)
[-20, -15, -10, -5, 0, 5, 10, 15, 20]
[-4.0, 5.0, 14.0, 23.0, 32.0, 41.0, 50.0, 59.0,
68.0]
```

- We probably would like to see the temperature in degrees celsius beside the corresponding temperature in degrees fahrenheit.
- This is easy to do if each row contains a temperature in degrees celsius and the corresponding temperature in degrees fahrenheit.

```
In [2]: temps = [[tC, tF] for tC, tF in
                zip(tempC, tempF)]
In [3]: print(temps)
[[-20, -4.0], [-15, 5.0], [-10, 14.0],
[-5, 23.0], [0, 32.0], [5, 41.0], [10, 50.0],
[15, 59.0], [20, 68.0]]
```

- This new table has 9 rows and 2 columns.
- If we display the rows of this new table we get:

```
In [1]: for row in temps:
            print("%6.1f  %6.1f" % (row[0], row[1]))
```

OR

```
In [2]: for tC, tF in temps:
            print("%6.1f  %6.1f" % (tC, tF))
```

```
 -20.0    -4.0
 -15.0     5.0
 -10.0    14.0
  -5.0    23.0
   0.0    32.0
   5.0    41.0
  10.0    50.0
  15.0    59.0
  20.0    68.0
```

- The default for displaying lists can be hard to read, the *pprint* function of the *pprint* module displays a list in a way that is easier to read.

```
In [1]: print(temps)
[[-20, -4.0], [-15, 5.0], [-10, 14.0], [-5,
23.0], [0, 32.0], [5, 41.0], [10, 50.0], [15,
59.0], [20, 68.0]]
In [2]: import pprint
In [3]: pprint.pprint(temps)
[[-20, -4.0],
 [-15, 5.0],
 [-10, 14.0],
 [-5, 23.0],
 [0, 32.0],
 [5, 41.0],
 [10, 50.0],
 [15, 59.0],
 [20, 68.0]]
```

Extracting Sub-lists or Slices

- Part of a list is known as a sub-list or slice.

```
In [1]: numbers = list(range(1, 15, 2))
In [2]: print numbers
[1, 3, 5, 7, 9, 11, 13]
In [3]: print(len(numbers))
7
In [4]: print(numbers[2:]) # elements at positions
                           # 2,3,4,5,6
[5, 7, 9, 11, 13]
In [5]: print(numbers[1:5]) # elements at positions
                            # 1,2,3,4
[3, 5, 7, 9]
In [6]: print(numbers[:5]) # elements at positions
                           # 0,1,2,3,4
[1, 3, 5, 7, 9]
In [7]: print(numbers[1:-1]) # all but first and last
                             # elements
[3, 5, 7, 9, 11]
```

- A slice has three parts: *start*, *stop* and *step,* just like a range does, separated by colons.
- *Start* and *stop* are positions in a list, *step* is the number of positions between the elements of the slice.

```
In [1]: numbers = list(range(1, 15, 2))
In [2]: print(numbers)
[1, 3, 5, 7, 9, 11, 13]
In [3]: print(numbers[::2])  # every second element
[1, 5, 9, 13]
In [4]: print(numbers[1:-1:3]) # every third element
[3, 9]
In [5]: print(numbers[-1::-1])  # reverse the order
                                # of the elements
[13, 11, 9, 7, 5, 3, 1]
```

```
In [1]: print(temps)
[[-20, -4.0], [-15, 5.0], [-10, 14.0],
[-5, 23.0], [0, 32.0], [5, 41.0], [10, 50.0],
[15, 59.0], [20, 68.0]]

In [2]: print(temps[6:]) # rows 6, 7, 8
[[10, 50.0], [15, 59.0], [20, 68.0]]

In [3]: print(temps[2:6]) # rows 2, 3, 4, 5
[[-10, 14.0], [-5, 23.0], [0, 32.0], [5, 41.0]]

In [4]: print(temps[2:6][1:3]) # rows 1, 2 of
                               # temps[2:6]
[[-5, 23.0], [0, 32.0]]
```

- A slice is a copy of part of a list, modifying the slice does not change the original list and vice-versa.

```
In [1]: print(numbers)
[1, 3, 5, 7, 9, 11, 13]
In [2]: odds = numbers[1:-1]
In [3]: print(odds)
[3, 5, 7, 9, 11]
In [4]: odds[2] = 0
In [5]: numbers[3] = 99
In [6]: print(odds)
[3, 5, 0, 9, 11]
In [7]: print(numbers)
[1, 3, 5, 99, 9, 11, 13]
```

- Write a program to print detailed sales information and the total sales for each salesperson. The information is stored in a table that has one row for each salesperson.
- Each row contains the salesperson's name and individual sales.

Name	Sale 1	Sale 2	Sale 3	Sale 4
Bob	$1500	$840	$1750	
Pat	$450	$900		
Fred	$990	$550	$275	$1200
Jane	$1330	$1075	$1100	

- Table Example: sales.py

```
sales = []
sales.append(['Bob', 1500., 840., 1750.])
sales.append(['Pat', 450., 900.])
sales.append(['Fred', 990., 550., 275., 1200.])
sales.append(['Jane', 1330., 1075., 1100.])

for person in sales:
    print('Sales person:', person[0])
    total = 0.
    for amount in person[1:]:
        print('Sale amount: $%7.2f' % amount)
        total += amount
    print('Total sales: $%7.2f\n' % total)
```

- The output from this program follows

```
Sales person: Bob
Sale amount: $1500.00
Sale amount: $ 840.00
Sale amount: $1750.00
Total sales: $4090.00

Sales person: Pat
Sale amount: $ 450.00
Sale amount: $ 900.00
Total sales: $1350.00

Sales person: Fred
Sale amount: $ 990.00
Sale amount: $ 550.00
Sale amount: $ 275.00
Sale amount: $1200.00
Total sales: $3015.00

Sales person: Jane
Sale amount: $1330.00
Sale amount: $1075.00
Sale amount: $1100.00
Total sales: $3505.00
```

- Alternate solution: sales1.py

```
sales = []
sales.append(['Bob', 1500., 840., 1750.])
sales.append(['Pat', 450., 900.])
sales.append(['Fred', 990., 550, 275., 1200.])
sales.append(['Jane', 1330., 1075., 1100.])

for person in sales:
    print('Sales for', person[0], '->', end='
')
    total = 0.
    for amount in person[1:]:
        print('$%.2f ' % amount, end=' ')
        total += amount
    print('Total: $%.2f\n' % total)
```

- The output from the program is:

```
Sales for Bob -> $1500.00  $840.00  $1750.00  Total: $4090.00
Sales for Pat -> $450.00  $900.00  Total: $1350.00
Sales for Fred -> $990.00  $550.00  $275.00  $1200.00  Total: $3015.00
Sales for Jane -> $1330.00  $1075.00  $1100.00  Total: $3505.00
```