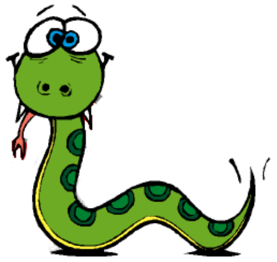
IDCE 302: Chapter 9

Tuples

Jie Tian, PhD

Clark University



Outline

- What is a tuple?
- Tuple operations (create, assign)
- Random numbers
- Classification example
- Sorting algorithm example

What is a tuple?

- In Python, a tuple is similar to a list except that **it is immutable**
- A tuple is defined within a pair of **parentheses**

```
>>> tuple = ('a', 'b', 'c', 'd', 'e')

>>> tuple[0] = 'A'  # not allowed!
TypeError: object doesn't support item
assignment
```

- To create a tuple with **a single element**, we have to include **the final comma**:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

- Without the comma, Python treats ('a') as a string in parentheses:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>

# list = ['a'] creates a single-element
list
```

Access Tuple Element

- Very similar to how you access list elements.

```
>>> tuple = ('a', 'b', 'c', 'd', 'e')

# get one element
>>> tuple[0]
'a'

# get a subset of a tuple
>>> tuple[1:3]
('b', 'c')
```

Assign Tuples to Variables

```
tupleUpper = ('A', 'B', 'C')
tupleLower = ('a', 'b', 'c')

print 'Original'
print tupleUpper
print tupleLower

tmp = tupleUpper
tupleCap=tupleLower
tupleSmall=tmp

print ‘After swapping'
print tupleUpper
print tupleLower
```

Output:

```
Original
('A', 'B', 'C')
('a', 'b', 'c')
After swapping
('a', 'b', 'c')
('A', 'B', 'C')
```

Tuples as Return Values

```
def swap(x, y):
    return y, x # it returns the two inputting parameters
                # in   a   reversed   order
a=('Hello', 'world', 2012)
b=(1, 3, 5, 7)
print a, '\n', b

a, b = swap(a,b)
print 'After executing a, b = swap(a,b)'
print a, '\n', b

# output
('Hello', 'world', 2012)
(1, 3, 5, 7)
After executing a, b = swap(a,b)
(1, 3, 5, 7)
('Hello', 'world', 2012)
```

```
def swap(x, y):
    return y, x

a=8
b=‘Mondays are busy.'
print a, '\n', b

a, b = swap(a,b)
print 'After executing a, b = swap(a,b)'
print a, '\n', b
```

Output:

```
8
Mondays are busy.
After executing a, b = swap(a,b)
Mondays are busy.
8
```

Random Numbers

```
import random  # random is a module

 def getRandomMean(iCount):
    sum=0.0
    for i in range(iCount):
       x = random.random()
       sum = sum + x
       print x
    return sum /float(iCount)

x=5
y=getRandomMean(x)
print 'The mean of ', x, 'random values is', y
```

```
Output (you may have different results):
0.281179112841
0.296812545904
0.72409395725
0.968349799157
0.140692783358
The mean of  5 random values is 0.482225639702
```

Creating a List of Random Numbers

```
import random

def randomList(n):
    s = [0] * n   #the use of list multiplication
    for i in range(n):
        s[i] = random.random()
    return s

n=5
print randomList(5)
```

```
Output:
[0.48015017371773094, 0.78567727850590852, 0.85662315185588034,
0.9387771241845918, 0.46579108529655644]
```

Can you create a tuple of random numbers in the same way?

Algorithms

- Algorithms design is very important to the quality of programs.
- Keep the following in mind:
  - **Readability**: easy to read /understand (for future debugging / enhancement & communication)
  - **Efficiency**: should be fast. Efficiency is an indicator of quality of algorithm
  - **Memory requirement**: memory availability is a constraint in programming.

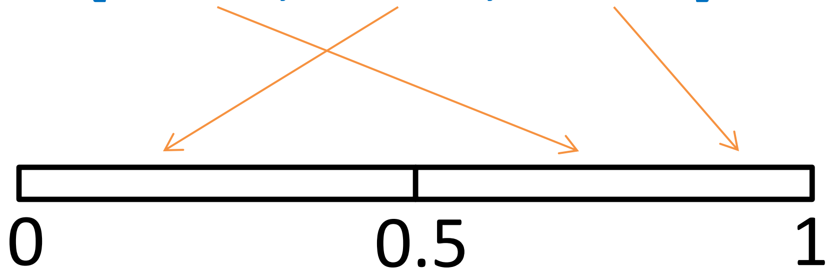
Classic Examples

Problem Set Up

- Say you have a list of random numbers
  [0.567, 0.213, 0.986]
- Then you want classify them into 2 classes with equal intervals (0—0.5, 0.5—1)
- Then you want to count how many falling in each class and get a list of the counts
- In this case, the expected result is
  [1, 2]
- Two examples on the next page

Input

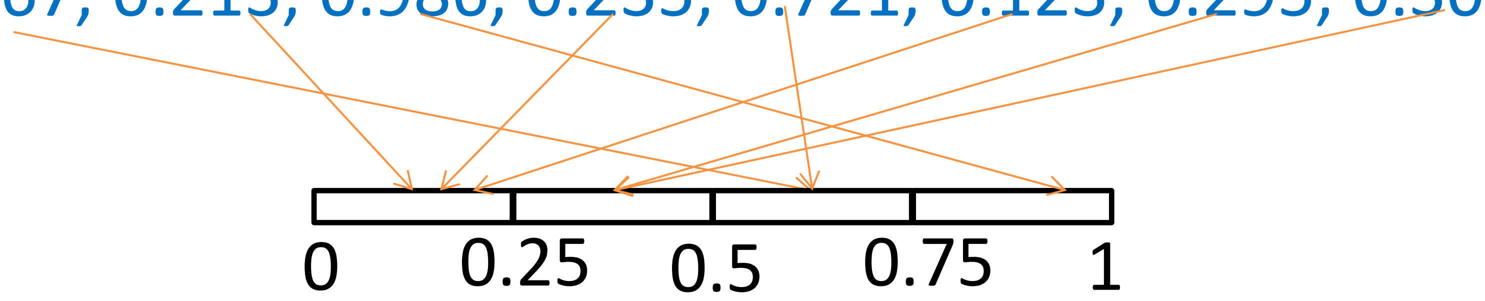
[0.567, 0.213, 0.986] 2



Output [1, 2]

Input

[0.567, 0.213, 0.986, 0.235, 0.721, 0.123, 0.295, 0.308] 4



Output [3, 2, 2, 1]

Think about this!

- Did you find that for each class (or bin), the lower bound can be determined using the bin width?
- For example, the lower bound for the 1st bin is 0 * binwidth; the lower bound for the 2nd bin is 1*binwidth, and so on.
- The upper bound for each bin is apparently equal the lower bound value + bin width

Code to solve the problem!

- We want to write a general function that can take a list and an integer number as arguments.
  - The list should be a list of random numbers that we are going to check.
  - The integer number should be the number of classes we decide.

Writing the function step by step

Step 1: Define the function

```
def classify(listRandom, intNumBins):
```

Note1: I named the function as **classify** but you have the freedom to name it something else.

Note2: I name the first argument as **listRandom** and the second as **intNumBins**. You can name them differently as long as you keep the names consistent in the function.

Step 2: Initialize the output

```
def classify(listRandom, intNumBins):
    listBinCount=[0]* intNumBins
```

Note: We know we want to return a list that has a length of intNumBins. So we initialize such a list of intNumBins 0s. We will of course change these 0s later in the code.

Step 3: Figure out the classwidth

```
def classify(listRandom, intNumBins):
    listBinCount=[0]* intNumBins
    floatBinWidth = 1.0 / intNumBins
```

Note: the class width is depending on the number of classes. It is 0.5 if 2 classes, 0.25 if 4 classes, 0.2 if 5 classes.

Step 4: Count for each class (or bin)

```
def classify(listRandom, intNumBins):
     listBinCount=[0]* intNumBins
      floatBinWidth = 1.0 / intNumBins
    for i  in  range(intNumBins):
       low = i * floatBinWidth
       high =  low + floatBinWidth
       listBinCount[i]= countBin(listRandom, low, high)
```

Note: Each round of the for loop is to count how many random numbers in the list fall into the bin.

1st round for the 1st bin, 2nd round for the 2nd bin, so on and so forth.

Step 5: finally, return the list of counts

```
def classify(listRandom, intNumBins):
     listBinCount=[0]* intNumBins
      floatBinWidth = 1.0 / intNumBins
    for i  in  range(intNumBins):
       low = i * floatBinWidth
       high =  low + floatBinWidth
       listBinCount[i]= countBin(listRandom, low, high)
    return listBinCount
```

Note1: Once we are done with the loop, we want to return the list of counts .

Note2: Are we missing anything still?

- Yes, we are still missing something because we called a function **`countBin`** that is not defined yet.

```
def countBin(inputRandomList, lowerbound, upperbound):
  intCount   = 0 # initialize
  for element in inputRandomList:
    if lowerbound <= element < upperbound :
      intCount = intCount + 1 #an element found in range, add 1
  return count # return the count
```

Are we all set?
Almost. Just missing the function producing a list of random numbers, which is easy to write.

Generating a list of random numbers

```
import random
def randomList(n):
    s=[0]*n # initialize
    for i in range(n):
      # replace one 0 with a random number a time
      s[i]=random.random()
    return s # return the list of random numbers
```

We are ready to put all the pieces together!!

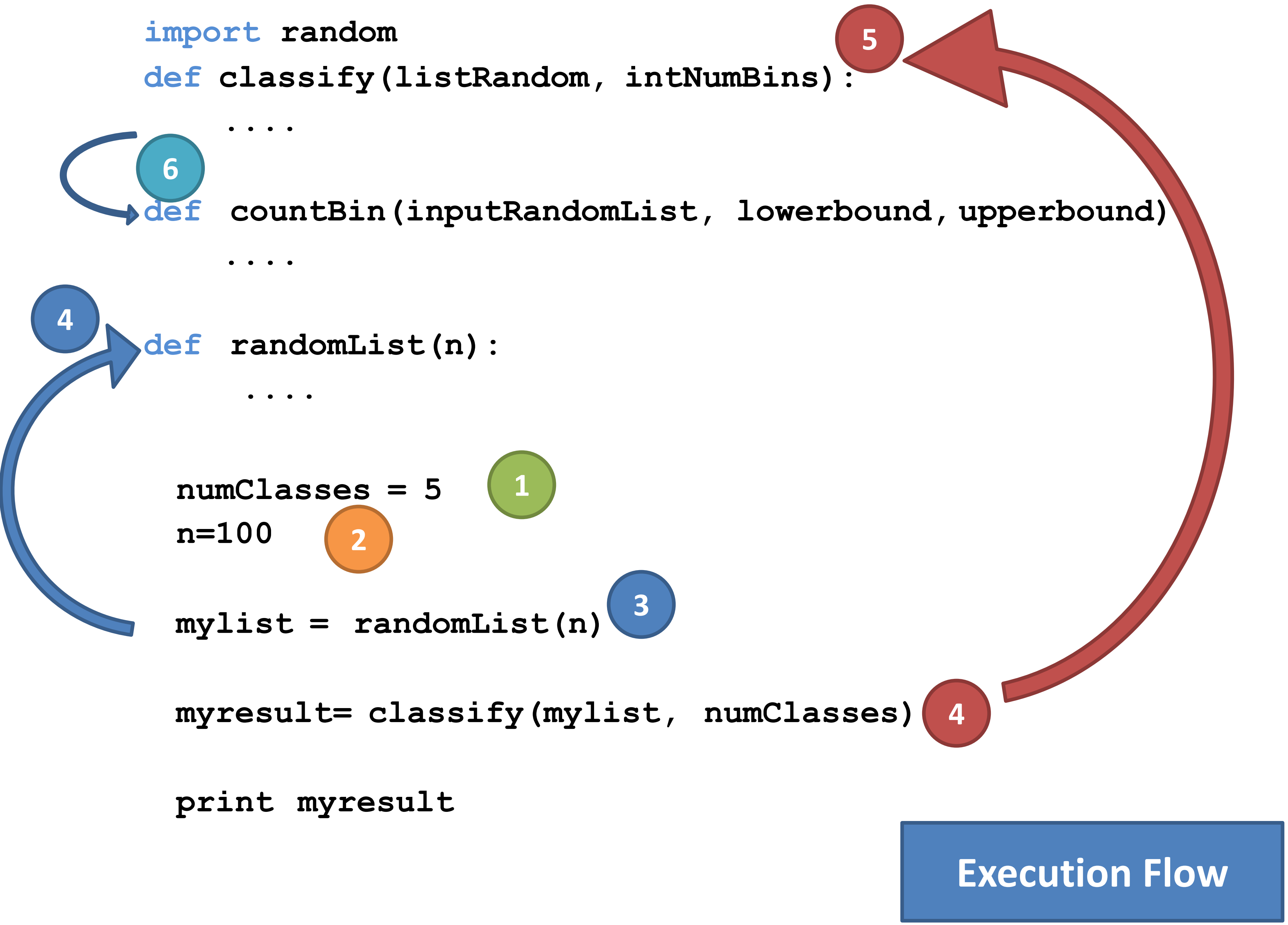
The 3 needed functions defined. Yeah..

```
import random
def classify(listRandom, intNumBins):
        listBinCount=[0]* intNumBins # initialize
        floatBinWidth = 1.0 / intNumBins # calculate width
    for  i  in  range(intNumBins):
         low = i * floatBinWidth  # calculate lower bound
         high =  low + floatBinWidth # calculate upper bound
         listBinCount[i]= countBin(listRandom, low, high) # count
    return listBinCount

def  countBin(inputRandomList, lowerbound, upperbound):
  intCount   = 0 # initialize
  for  element in inputRandomList:
    if lowerbound <= element < upperbound  :
      intCount = intCount + 1 #an element found in range, add 1
    return count # return the count

def  randomList(n):
    s=[0]*n # initialize
       for   i  in  range(n):
      # replace one 0 with a random number a time
        s[i]=random.random()
    return s # return the list of random numbers
```

Can use them now

```
# Assign values
numClasses = 5 # 5 classes or bins in this case
n=100 # let python generate 100 random numbers in this case

# call the randomList function to generate a list of
# random numbers, 100 of them in this case
mylist =  randomList(n)

# plug in the list generated above and the number of bins
# which is 5 in this case
myresult= classify(mylist, numClasses)

# finally just print the returned list with 5 elements in
# this case
print myresult
```

Note: Next time, you can easily perform classifying 500 random numbers into 10 classes and count, by assigning 10 to numClasses and 500 to n. Nothing else needs to be changed!



```
import random
def classify(listRandom, intNumBins):
    ....

def  countBin(inputRandomList, lowerbound, upperbound)
    ....

def  randomList(n):
     ....

   numClasses = 5
   n=100

   mylist =  randomList(n)

   myresult= classify(mylist,  numClasses)

   print  myresult
```

Sorting Algorithm (Step by Step)

Step 0. Problem set up

- We need to write a function to sort a list of numbers.
- The function takes a <u>list of numbers</u>.
- The functions sorts the list.

Step 1. Strategy?

- Starting from the first element of an input list, <u>compare</u> its value with the value of <u>each element after it</u>.
- The element with the "smallest" value is switched with the element being checked.

Step 2. Loop through the list

```
def  sortList(list):
   i=0 # initialize
   # loop through the list
    # len(list) gets the list length
  while i < len(list):
        … # Here to write the comparison and
          # switching!!!
```

Step 4. Initialize before comparing

```
def  sortList(list):
    i=0 # initialize
    # loop through the list, len(list) gets the list length
  while i < len(list):
      # suppose the element being checked is the smallest for now
      minVal = list[i]# use minVal to hold the Min value so far
      minIndex = i # use minIndex to hold the corresponding index
```

initialization

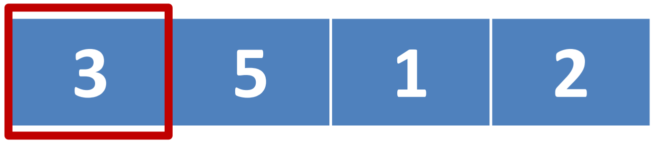
Step 5. Compare!!

```
def  sortList(list):
     i=0 # initialize
     # loop through the list, len(list) gets the list length
   while i < len(list):
          # suppose the element being checked is the smallest
          # compared to all the elements after it.
          minVal = list[i]# use minVal to hold the Min value so far
          minIndex = i # use minIndex to hold the corresponding index

          j = i + 1 # Only compare to the "after" elements
          # loop through the "after" elements, that's why j = i +1
          while j < len(list):
                 if minVal > list[j]: # if a smaller value found
                    minVal = list [j] # assign the value to minVal, updating.
                    minIndex = j # assign the index to minIndex, updating.

                 j = j + 1

          # Once the loop is done, minVal holds the smallest value among all
          # the comparisons made. minIndex holds the corresponding index
```

Step 6. Switch!!

```
def  sortList(list):
    i=0 # initialize
    # loop through the list, len(list) gets the list length
  while i < len(list):
        # suppose the element being checked is the smallest
        # compared to all the elements after it.
        minVal = list[i]# use minVal to hold the Min value so far
        minIndex = i # use minIndex to hold the corresponding index
        j = i + 1 # Only compare to the "after" elements
        # loop through the the "after" elements, that's why j = i +1
        while j < len(list):
            if minVal > list[j]: # if a smaller value found
              minVal = list [j] # assign the value to minVal, updating.
              minIndex = j # assign the index to minIndex, updating.
            j = j + 1
        # Switch the element being checked and the one found with lowest
        # value
        temp = list[i]
        list[i] = list[minIndex] # remember what is minIndex?
        list[minIndex]=temp

        i = i+1
```

Demo



list = [3,5,1,2]

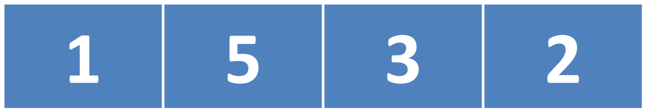
```
i: 0
minVal: 3
minIndex: 0
```

```
j: 1
minVal: 3
minIndex: 0
```

```
j: 2
minVal: 1
minIndex: 2
```

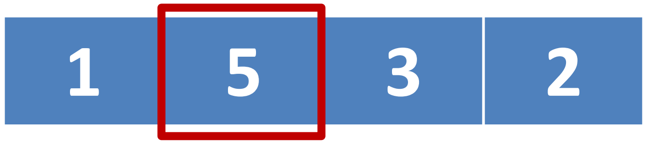
```
j: 3
minVal: 1
minIndex: 2
```

```
temp = list [0]
list [0] = list [2]
List [2] = temp
```



list = [1,5,3,2]

Demo

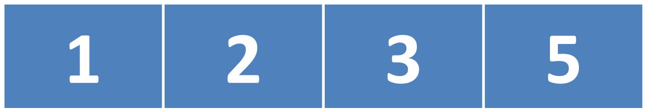


`list = [1,5,3,2]`

```
i: 1
minVal: 5
minIndex: 1
```

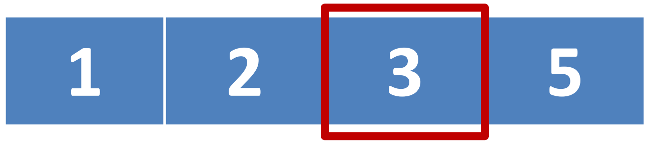
```
j: 2                    j: 3
minVal: 3               minVal: 2
minIndex: 2             minIndex: 3
```

```
temp = list [1]
list [1] = list [3]
list [1] = temp
```



`list = [1,2,3,5]`

Demo

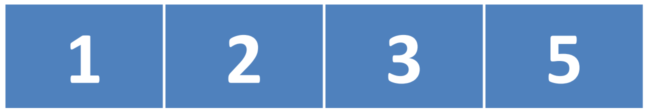


```
list = [1,2,3,5]
```

```
i: 2
minVal: 3
minIndex: 2
```

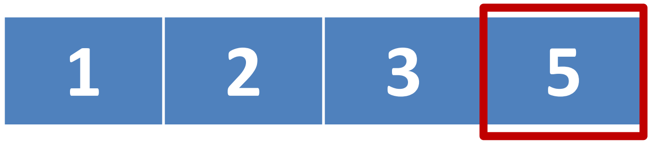
```
j: 3
minVal: 3
minIndex: 2
```

```
temp = list [2]
list [2] = list [2]
List [2] = temp
```



```
list = [1,2,3,5]
```

Demo

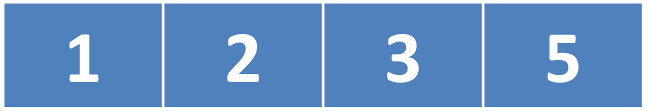


```
list = [1,2,3,5]
```

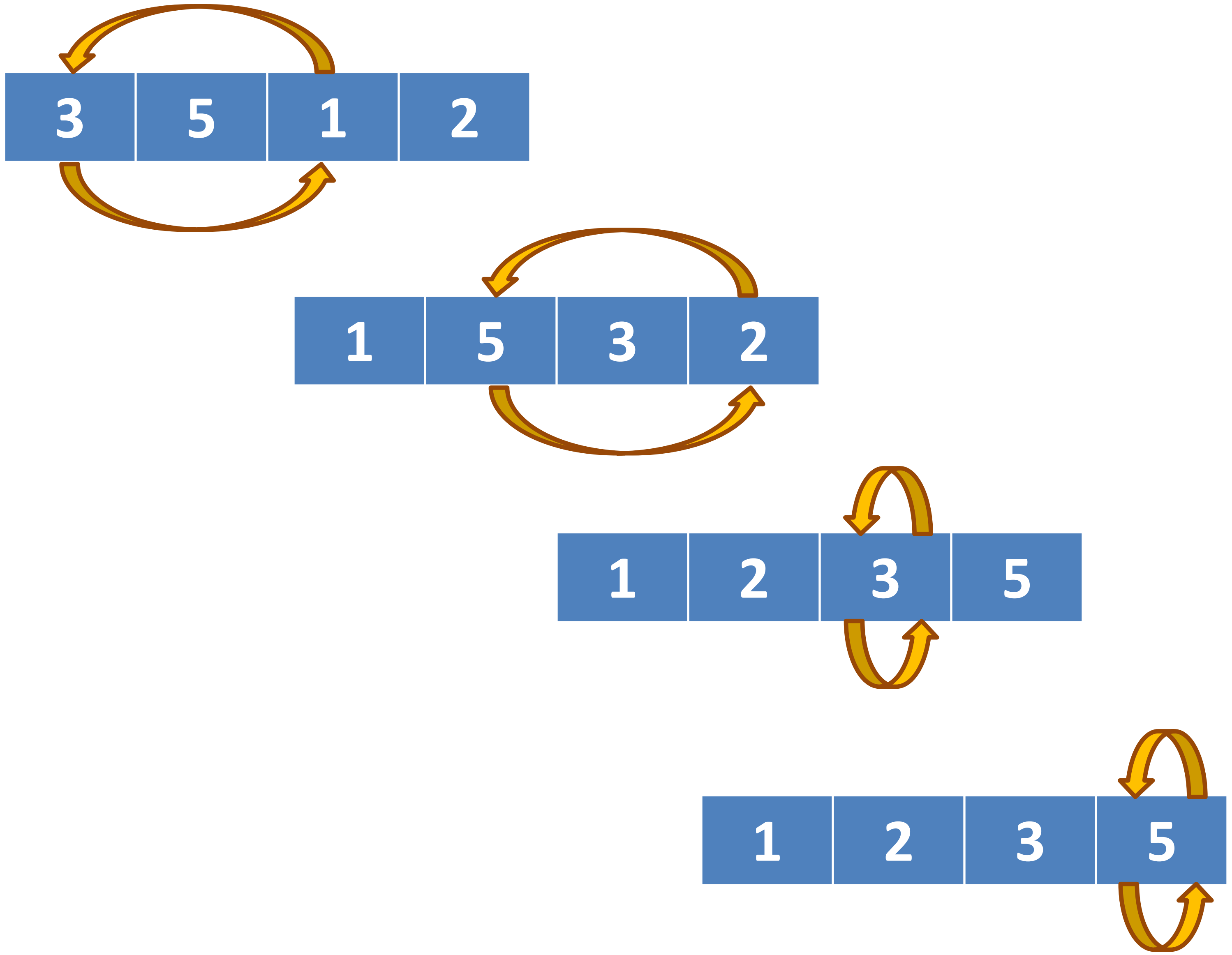
```
i: 3
minVal: 5
minIndex: 3
```

```
j: 4 (it's beyond the range, so the nested while loop skipped)
```

```
temp = list [3]
list [3] = list [3]
List [3] = temp
```



```
list = [1,2,3,5]
```

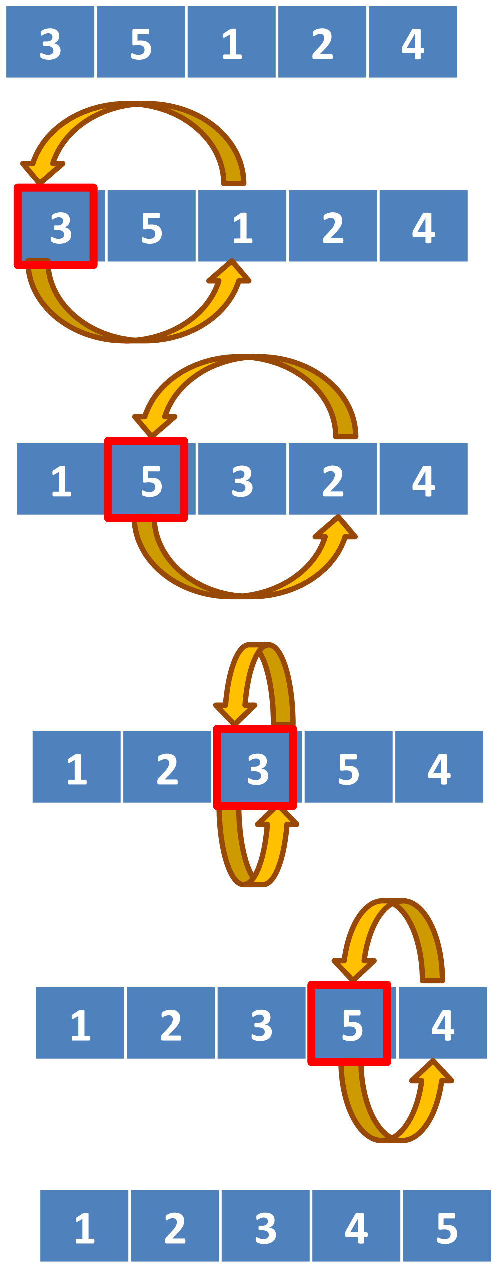


3 5 1 2
1 5 3 2
1 2 3 5
1 2 3 5

Sample Code: Sorting Algorithm

```
def  sortList(list):
     i=0
     while  i  <  len(list):
          minVal =  list[i]
          minIndex  =  i
          j=i+1
          while  j  <  len(list):
              if  minVal>list[j]:
                  minVal =  list[j]
                  minIndex  =  j
             j  =  j+1
          temp =  list[i]
          list[i]  =  list[minIndex]
          list[minIndex]=temp
          i  =  i+1
```

```
n=5
myList=  randomList(n)
print  "Before  sorting:"
print  myList
sortList(myList)
print  "After  sorting:"
print  myList
```



Built-in functions: sort ()   sorted()

Summary

- Tuple is similar to a list, but it cannot be modified.
- Random module for using random numbers.
- **return x, y returns a tuple (x, y)**
- Understand the classification example and the sorting algorithm example.

```
>>> from operator import  itemgetter
>>> sorted(sequence, key = itemgetter(0), reverse = False)
[(0, 4, 'd'), (1, 3, 'e'), (2, 2, 'a'), (3, 1, 'b'), (4, 0, 'c')]
```