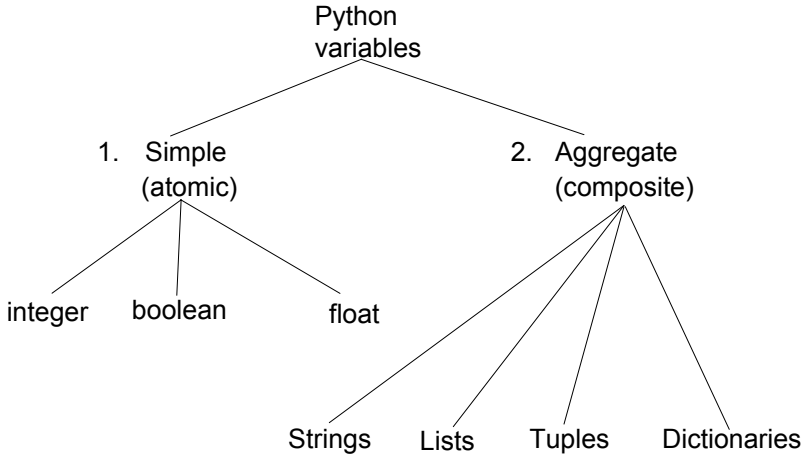
Composite Types

You will learn in this section of notes how to create single and generic instances of non-homogeneous composite types that are used for different scenarios.

James Tam

Types Of Variables



James Tam

Small Example Programs Using Strings

- They can be found online under the following names
  - string1.py (passing a whole string to a function)
  - string2.py (indexing the parts of a string)
  - string3.py (demonstrating the immutability of strings)
  - string4.py (string slicing)
  - string5.py (strings as sets, test for inclusion using 'in')
  - string6.py (strings that are repetitive sequence)
  - string7.py (using string functions: converting string input to numerical)
  - string8.py (using string functions that return modified versions of a string)
  - string9.py (string search functions)
- All the examples will be located in UNIX under:
  /home/231/examples/composites

James Tam

String

- Strings are just a series of characters (e.g., alpha, numeric, punctuation etc.)
- A string can be treated as one entity.

```
def fun (aString):
    print aString

# MAIN
aString = "Goodbye cruel world!"
fun (aString)
```

- Or the individual elements (characters) can be accessed via an index.
  - Note: A string with 'n' elements has an index from 0 to (n-1)

```
# MAIN
aString = "hello"
print aString[1],
print aString[4],
```

James Tam

Strings Are Immutable

- Even though it may look a string can change they actually cannot be edited.

```
# MAIN
aString = "good-bye"
print aString
aString = "hello"
print aString
aString[0] = "G“          # Error
```


Substring Operations

- Sometimes you may wish to extract out a portion of a string.
  - E.g., Extract out “James” from “James T. Kirk, Captain”
- This operation is referred to as a ‘substring’ operation in many programming languages.
- There are two implementations of the substring operation in Python:
  - String slicing
  - String splitting

String Slicing

- Slicing a string will return a portion of a string based on the indices provided
- The index can indicate the start and end point of the substring.
- **Format**:
  string_name [*start_index* : *end_index*]

- **Example**:

```
aString = "abcdefghij"
print aString
temp = aString [2:5]
print temp
temp = aString [:5]
print temp
temp = aString [7:]
print temp
```

James Tam

String Splitting

- Divide a string into portions with a particular character determining where the split occurs.
  - The string "The cat in the hat" could be split into individual words
  - "The" "cat" "in" "the" "hat"
- **Format**:
  string_name.split ('<character used in the split')

- **Examples**:

```
aString = "man who smiles"
one, two, three = aString.split()   # Default character is a space
print one
print two
print three
aString = "Tam, James"
last, first = aString.split(',')
print first, last
```

James Tam

Strings Can Be Conceptualized As Sets

- The 'in' and 'not in' operations can be performed on a string.
- Branching

```
passwords = "aaa abc password xxx"
password = raw_input ("Password: ")
if password in passwords:
   print "You entered an existing password, enter a new one“
```

- Looping (iterating through the elements)

```
sentence = "hihi there!"
for temp in sentence:
   sys.stdout.write(temp)
```

James Tam

Repetitive Strings

- A string with a number of repeated characters can be initialized in a number of ways.

```
aString = “xxxxxxxx”
aString = “hi!” * 5
```

James Tam

String Testing Functions[1]

- These functions test a string to see if a given condition has been met and return either "True" or "False" (Boolean).
- **Format**:
  string_name.*function_name* ()

1 These functions will return false if the string is empty (less than one character).

James Tam

String Testing Functions (2)

Boolean Function	Description
isalpha ()	Only true if the string consists only of alphabetic characters.
isdigit ()	Only returns true if the string consists only of digits.
isalnum ()	Only returns true if the string is composed only of alphabetic characters or numeric digits.
islower ()	Only returns true if the alphabetic characters in the string are all lower case.
isspace ()	Only returns true if string consists only of whitespace characters (" ", "\n", "\t")
isupper ()	Only returns true if the alphabetic characters in the string are all upper case.

James Tam

Applying A String Testing Function

```
# MAIN
ok = False
while (ok == False):
   temp = raw_input ("Enter numbers not characters: ")
   ok = temp.isdigit()
   if (ok == False):
      print temp, "is not a number"
   else:
      print "done"
num = int (temp)
num = num + num
print num
```

James Tam

Functions That Modify Strings

- These functions return a modified version of an existing string (leaves the original string intact).

Function	Description
lower ()	Returns a copy of the string with all the alpha characters as lower case (non-alpha characters are unaffected).
upper ()	Returns a copy of the string with all the alpha characters as upper case (non-alpha characters are unaffected).
strip ()	Returns a copy of the string with all leading and trailing whitespace characters removed.
lstrip ()	Returns a copy of the string with all leading (left) whitespace characters removed.
rstrip ()	Returns a copy of the string with all trailing (right) whitespace characters removed.
lstrip (char)	Returns a copy of the string with all leading instances of the character parameter removed.
rstrip (char)	Returns a copy of the string with all trailing instances of the character parameter removed.

James Tam

Example Uses Of Functions That Modify Strings

```
aString = "talk1! AbouT"
print aString
aString = aString.upper ()
print aString

aString = "xxhello there"
print aString
aString = aString.lstrip ('x')
print aString
aString = "xxhellx thxrx"
aString = aString.lstrip ('x')
print aString
```

James Tam

Functions To Search Strings

Function	Description
endswith (substring)	A substring is the parameter and the function returns true only if the string ends with the substring.
startswith (substring)	A substring is the parameter and the function returns true only if the string starts with the substring.
find (substring)	A substring is the parameter and the function returns the lowest index in the string where the substring is found (or -1 if the substring was not found).
replace (oldstring, newstring)	The function returns a copy of the string with all instances of 'oldstring' replace by 'newstring'

James Tam

Examples Of Functions To Search Strings

```
temp = raw_input ("Enter a sentence: ")
if not ((temp.endswith('.')) or (temp.endswith('!')) or (temp.endswith ('?'))):
    print "Not a sentence"

temp = "XXabcXabcabc"
index = temp.find("abc")
print index

temp = temp.replace("abc", "Abc")
print temp
```

James Tam

List

- In many programming languages a list is implemented as an array.
- Python lists have many of the characteristics of the arrays in other programming languages but they also have many other features.
- This first section will talk about the features of lists that are largely common to arrays.

James Tam

Example Problem

- Write a program that will track the percentage grades for a class of students. The program should allow the user to enter the grade for each student. Then it will display the grades for the whole class along with the average.

James Tam

Why Bother With Composite Types?

- The full example can be found in UNIX under:
  /home/231/examples/composites/classList1.py

```
CLASS_SIZE = 5
stu1 = 0
stu2 = 0
stu3 = 0
stu4 = 0
stu5 = 0
total = 0
average = 0

stu1 = input ("Enter grade for student no. 1: ")
stu2 = input ("Enter grade for student no. 2: ")
stu3 = input ("Enter grade for student no. 3: ")
stu4 = input ("Enter grade for student no. 4: ")
stu5 = input ("Enter grade for student no. 5: ")
```

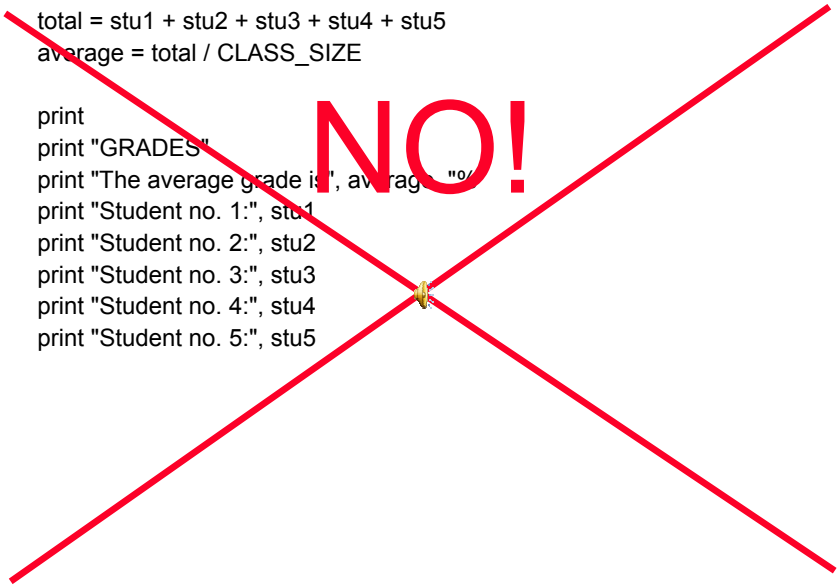
James Tam

Why Bother With Composite Types? (2)

```
total = stu1 + stu2 + stu3 + stu4 + stu5
average = total / CLASS_SIZE

print
print "GRADES"
print "The average grade is", average, "%"
print "Student no. 1:", stu1
print "Student no. 2:", stu2
print "Student no. 3:", stu3
print "Student no. 4:", stu4
print "Student no. 5:", stu5
```

James Tam

Why Bother With Composite Types? (2)




```
total = stu1 + stu2 + stu3 + stu4 + stu5
average = total / CLASS_SIZE

print
print "GRADES"
print "The average grade is", average, "%"
print "Student no. 1:", stu1
print "Student no. 2:", stu2
print "Student no. 3:", stu3
print "Student no. 4:", stu4
print "Student no. 5:", stu5
```

James Tam

What Were The Problems With The Previous Approach?

- Redundant statements.
- Yet a loop could not be easily employed given the types of variables that you have seen so far.

James Tam

What's Needed

- A composite variable that is a collection of another type.
  - The composite variable can be manipulated and passed throughout the program as a single entity.
  - At the same time each element can be accessed individually.
- What's needed…an array / list!

James Tam

Creating A List (No Looping)

- This step is mandatory in order to allocate memory for the array.
- Omitting this step (or the equivalent) will result in a syntax error.
- **Format:**

  <*array_name*> = [<*value 1*>, <*value 2*>, ... <*value n*>]

Example:

```
percentages = [0.0, 0.0, 0.0, 0.0, 0.0]
letters = ['A', 'A', 'A']
names = ["James Tam", "Stacey Walls", "Jamie Smyth"]
```


Creating A List (With Loops)

- Step 1: Create a variable that is a reference to the list
- **Format:**

  <*list name*> = []

- **Example:**

```
classGrades = []
```

Creating A List (With Loops: 2)

•Step 2: Initialize the list with the elements

•**General format:**

- Within the body of a loop create each element and then append the new element on the end of the list.

•**Example:**

```
for i in range (0, 5, 1):
    classGrades.append (0)
```

James Tam

Revised Version Using A List

•The full example can be found in UNIX under:
/home/231/examples/composites/classList2.py

```
CLASS_SIZE = 5
i = 0
total = 0
average = 0
classGrades = []

for i in range (0, CLASS_SIZE, 1):
    classGrades.append(0)
```

James Tam

Revised Version Using A List (2)

```
for i in range (0, CLASS_SIZE, 1):
    print "Enter grade for student no.", (i+1), ":",
    classGrades[i] = input ()
    total = total + classGrades[i]
average = total / CLASS_SIZE

print
print "GRADES"
print "The average grade is", average, "%"
for i in range (0, CLASS_SIZE, 1):
     print "Student no.", (i+1)
```


Printing Lists

- Although the previous example stepped through each element of the list in order to display it's contents onscreen if you want to quickly check the contents (and not worry about details like formatting) then you can simply use a print statement as you would with any other variable.

Example:

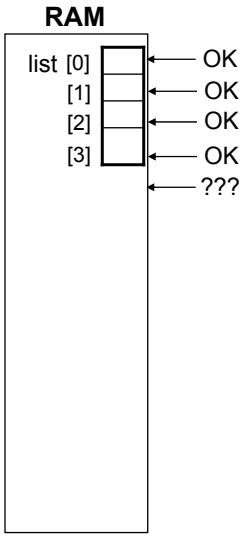
```
print classGrades
```

Output:

```
[10, 20, 30, 40, 50]
```

Take Care Not To Exceed The Bounds Of The List

```
list = [0, 1, 2, 3]
for i in range (0, 4, 1):
   print list [i],

print
print list [4]
```



James Tam

One Way Of Avoiding An Overflow Of The List

- Use a constant in conjunction with the list.

```
SIZE = 100
```

- The value in the constant controls traversals of the list

```
for i in range (0, SIZE, 1):
    myList [i] = raw_input ("Enter a value:")

for i in range (0, SIZE, 1):
   print myList [i]
```

James Tam

One Way Of Avoiding An Overflow Of The List

- Use a constant in conjunction with the list.
  SIZE = **100000**

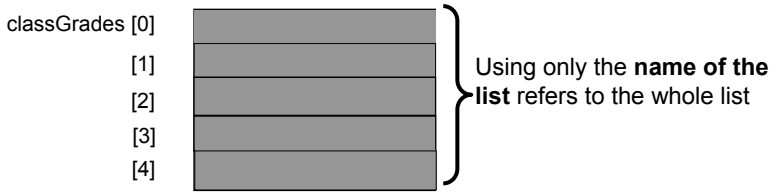
- The value in the constant controls traversals of the list

```
for i in range (0, SIZE, 1):
    myList [i] = raw_input ("Enter a value:")

for i in range (0, SIZE, 1):
    print myList [i]
```

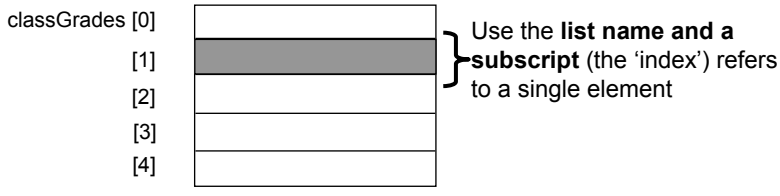
James Tam

Accessing Data In The List

- To manipulate an array you need to first indicate which list is being accessed
  - Done via the name of the list e.g., "print classGrades"



- If you are accessing a single element, you need to indicate which element that you wish to access.
  - Done via the list index e.g., "print classGrades[1]"



James Tam

Important Things To Keep In Mind

- (What you should now): Lists are a composite type that can be decomposed into other types.
- Other important points:
  - Copying lists
  - Passing lists as parameters

James Tam

Copying Lists

- A list variable is not actually a list!
- Instead that list variable is actually a reference to the list.
- (This is important because if you use the assignment operator to copy from list to another you will end up with only one list).
- Example:
  - The full example can be found in UNIX under: /home/231/examples/composites/copy1.py

```
list1 = [1,2]
list2 = [2,1]
print list1, list2

list1 = list2
print list1, list2

list1[0] = 99
print list1, list2
```

James Tam

Copying Lists (2)

- To copy the elements of one list to another a loop is needed to copy each successive elements.
- Example:
  - The full example can be found in UNIX under: /home/231/examples/composites/copy2.py

```
list1 = [1,2,3,4]
list2 = []

for i in range (0, 4, 1):
   list2.append(list1[i])

print list1, list2
list1[1] = 99
print list1, list2
```

James Tam

Passing Lists As Parameters

- Unlike what you've seen with parameter passing so far, modifying a list that's been passed as a parameter to a function *may* modify the original list.
- It all depends upon how the list is accessed in the function.
- When a list is created the variable is not actually a list but only a reference to the list.
- When the reference is passed as a parameter to a function another reference also refers to the list.

James Tam

Original List Is Changed

- Passing lists into functions is done using a different mechanism
  - When a list is passed into the function a local reference refers to the original list.
  - Example:
  - The full example can be found in UNIX under: /home/231/examples/composites/parameter1.py

```
def fun (list):
  list[0] = 99
  print list


def main ():
  list = [1,2,3]
  print list
  fun (list)
  print list

main ()
```

James Tam

Original List Is Unchanged

- If the local reference is assigned to another list then it will obviously no longer refer to the original list.
- (Effect: changes made via the local reference will change the local list and not the original that was passed into the function).
- Example:
- The full example can be found in UNIX under: /home/231/examples/composites/parameter2.py

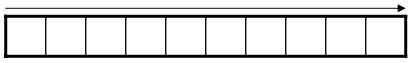
```
def fun (list):
  list = [3,2,1]
  print list


def main ():
  list = [1,2,3]
  print list
```

James Tam

When To Use Lists Of Different Dimensions

- Determined by the data – the number of categories of information determines the number of dimensions to use.
- Examples:
- (1D array)
  - Tracking grades for a class
  - Each cell contains the grade for a student i.e., grades[i]
  - There is one dimension that specifies which student's grades are being accessed

One dimension (which student)



- (2D array)
  - Expanded grades program
  - Again there is one dimension that specifies which student's grades are being accessed
  - The other dimension can be used to specify the lecture section

James Tam

When To Use Lists Of Different Dimensions (2)

- (2D list continued)

Student

Lecture section

	First student	Second student	Third student	…
L01				
L02				
L03				
L04				
L05				
:				
L0N				

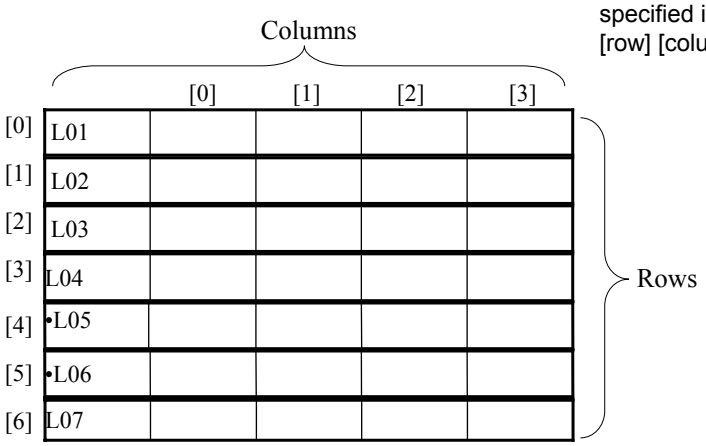
James Tam

When To Use Lists Of Different Dimensions (3)

- (2D list continued)
- Notice that each row is merely a 1D list
- (A 2D list is a list containing rows of 1D lists)

Important:

List elements are specified in the order of [row] [column]



Creating And Initializing A Multi-Dimensional List In Python

General structure

```
<array_name> = [ [<value 1>,  <value 2>, ... <value n>],
                 [<value 1>,  <value 2>, ... <value n>],
                       :          :          :
                       :          :          :
                 [<value 1>,  <value 2>, ... <value n>] ]
```

Rows

Columns

Creating And Initializing A Multi-Dimensional List In Python (2)

Example:

```
matrix = [ [0, 0, 0],
           [1, 1, 1],
           [2, 2, 2],
           [3, 3, 3]]

for r in range (0, 4, 1):
    for c in range (0, 3, 1):
        print matrix [r][c],
    print
```


Creating And Initializing A Multi-Dimensional List In Python (3)

- **General structure (Using loops):**
- Create a variable that refers to a 1D list. The outer loop traverses the rows. Each iteration of the outer loop creates a new 1D list. Then the inner loop traverses the columns of the newly created 1D list creating and initializing each element in a fashion similar to how a single 1D list was created and initialized.

- **Example (Using loops):**

```
aGrid = []                      # Create a reference to the list
for r in range (0, 3, 1):       # Outer loop runs once for each row
    aGrid.append ([])           # Create a row (a 1D list)
    for c in range (0, 3, 1):   # Inner loop runs once for each column
        aGrid[r].append (" ")   # Create and initialize each element (1D list)
```

Example 2D List Program: A Character-Based Grid

•The full example can be found in UNIX under:
/home/231/examples/composites/grid.py

```
import sys
import random

MAX_ROWS = 4
MAX_COLUMNS = 4
NO_COMBINATIONS = 10
```


A Character-Based Grid (2)

```
def generateElement (temp):
    anElement = '?'
    if (temp >= 1) and (temp <= 6):
        anElement = ' '
    elif (temp >= 7) and (temp <= 9):
        anElement = '*'
    elif (temp == 10):
        anElement = '.'
    else:
        print "<< Error with the random no. generator.>>"
        print "<< Value should be 1-10 but random value is ", temp
        anElement = '!'
    return anElement
```

A Character-Based Grid (3)

```
def initialize (aGrid):
    for r in range (0, MAX_ROWS, 1):
        for c in range (0, MAX_COLUMNS, 1):
            temp = random.randint (1, NO_COMBINATIONS)
            aGrid[r][c] = generateElement (temp)
```

James Tam

A Character-Based Grid (4)

```
def display (aGrid):
    for r in range (0, MAX_ROWS, 1):
        for c in range (0, MAX_COLUMNS, 1):
            sys.stdout.write(aGrid[r][c])
        print


def displayLines (aGrid):
    for r in range (0, MAX_ROWS, 1):
        print " - - - -"
        for c in range (0, MAX_COLUMNS, 1):
            sys.stdout.write ('|')
            sys.stdout.write (aGrid[r][c])
        print '|'
    print " - - - -"
```

James Tam

A Character-Based Grid (5)

• **# MAIN FUNCTION**

```
def main ():
    aGrid = []
    for r in range (0, MAX_ROWS, 1):
        aGrid.append ([])
        for c in range (0, MAX_COLUMNS, 1):
            aGrid[r].append (" ")

    initialize(aGrid)
    print "Displaying grid"
    print "==============="

    display (aGrid)
    print
    print "Displaying grid with bounding lines"
    print "==================================="
    displayLines (aGrid)
```


List Elements Need Not Store The Same Data Type

• What if different types of information needs to be tracked in the list?

Example, storing information about a client:

- Name …series of characters
- Phone number …numerical or character
- Email address …series of characters
- Total purchases made …numerical or character

Non-Homogeneous Lists

- If just a few clients need to be tracked then a simple list can be employed:

```
firstClient = ["James Tam"
               "(403)210-9455",
               "tamj@cpsc.ucalgary.ca",
                0]
```


Non-Homogeneous Lists (2)

- (Or as a small example)

```
def display (firstClient):
    print "DISPLAYING CLIENT INFORMATION"
    print "----------------------------"
    for i in range (0, 4, 1):
        print firstClient [i]

# MAIN
firstClient = ["James Tam"
               "(403)210-9455",
               "tamj@cpsc.ucalgary.ca",
                0]
display (firstClient)
```

Non-Homogeneous Lists (3)

- If only a few instances of the composite type (e.g., "Clients") need to be created then multiple instances single lists can be employed.

```
firstClient = ["James Tam"
               "(403)210-9455",
               "tamj@cpsc.ucalgary.ca",
               0]

secondClient = ["Peter Griffin"
                "(708)123-4567",
                "griffinp@familyguy.com",
                100]
```


Small Example Programs Using Lists

- The examples can be found in UNIX under /home/231/examples/composites/
  - list1.py (concatenation and repetition)
  - list2.py (membership)

Some List Operations

Operation name	Operator	Description
Indexing	[]	Access a list element
Concatenation	+	Combine lists
Repetition	*	Concatenate a repeated number of times
Membership	in	Query whether an item is a member of a list
Membership	not in	Query whether an item is not a member of a list
Length	len	Return the number of items in a list
Slicing	[:]	Extract a part of a list

James Tam

Examples: Concatenation And Repetition

```
list1 = [1, 2.0, "foo"]
list2 =  [[1,2,3], "salam"]
print list1
print list2
list1 = list1 * 2
print list1
list3 = list1 + list2
print list3
```

James Tam

Examples: Membership

```
print "Example 1: "
recall_list = ["vpn123", "ncc1946", "gst7"]
item = raw_input ("Product code: ")
if item in recall_list:
    print "Your product was on the recall list, take it back"
else:
    print "You're safe"
print

print "Example 2:"
days = ["Sun", "Mon", "Tue", "Wed", "Thur", "Fri", "Sat"]
for temp in days:
    print temp
```

James Tam

Some Useful List Operations

Operation	Format	Description
Append	list_name.append (item)	Adds a new item to the end of the list
Insert	list_name.insert (i, item)	Inserts a new item at index 'i'
Sort	list_name.sort ()	Sorts from smallest to largest
Reverse	list_name.reverse ()	Reverses the current order of the list
Count	list_name.count (item)	Counts and returns the number of occurrences of the item

James Tam

Tuples

- Much like a list, a tuple is a composite type whose elements can consist of any other type.
- Tuples support many of the same operators as lists such as indexing.
- However tuples are immutable.
- Tuples are used to store data that should not change.

James Tam

Creating Tuples

- **Format**:
  tuple_name = (*value*^{1}, *value*^{2}...*value*^{n})

- **Example**:
  ```
  tup = (1,2,"foo",0.3)
  ```

James Tam

A Small Example Using Tuples

- This example can be found online in UNIX under: /home/231/examples/composites/tuples1.py

```
tup = (1,2,"foo",0.3)
print tup
print tup[2]
tup[2] = "bar"
```

Error:
"TypeError: object does not support item assignment"

James Tam

Function Return Values

- Although it appears that functions in Python can return multiple values they are in fact consistent with how functions are defined in other programming languages.
- Functions can either return zero or *exactly one value* only.
- Specifying the return value with brackets merely returns one tuple back to the caller.

```
def fun ():
  return (1,2,3)
```

Returns: A tuple with three elements

```
Def fun (num):
  if (num > 0):
    print "pos"
    return
  elif (num < 0):
    print "neg"
    return
```

Nothing is returned back to the caller

James Tam

Dictionaries

- A special purpose composite type that maps keys (which can be any immutable type) to a value (like lists it can be any value).
- The keys can be used to later lookup information about the value e.g., looking up the definition for a word in a dictionary.

James Tam

Small Example Programs Using Dictionaries

- The examples can be found online in UNIX under:
  /home/231/examples/composites/
  - dictionary1.py (creating dictionaries)
  - dictionary2.py (deleting entries from the dictionary, checking for membership)

James Tam

Creating A Small Dictionary

- **Format** (defining the entire dictionary all at once)
  <*dictionary_name*> = {*key*[1]:*value*[1], *key*[2]:*value*[2]...*key*[n]:*value*[n]}

- **Example**: (defining the entire dictionary all at once)
  dict = {"one":"yut", "two":"yee", "three":"saam"}

James Tam

Creating A Large Dictionary

- **Format**:
  - *dictionary_name* = {}
  - *dictionary_name* [*key*[1]] = value[1]
  - *dictionary_name* [*key*[2]] = value[2]
  - : : :
  - *dictionary_name* [*key*[n]] = value[n]

- **Example**:
  dict = {}
  dict ["word1"] = ["Dictionary definition for word1"]
  dict ["word2"] = ["Dictionary definition for word2"]

James Tam

Examples Of Creating Dictionaries

```
dict = {}
dict ["word1"] = ["Dictionary definition for word1"]
dict ["word2"] = ["Dictionary definition for word2"]
dict ["word3"] = ["Dictionary definition for word3"]
temp = raw_input ("Enter dictionary definition for word4: ")
dict ["word4"] = [temp]
print dict

dict = {"one" : "yut", "two" : "yee", "three" : "saam"}
print dict
word = raw_input ("Enter word to translate: ")
print "English:", word, "\t", "Chinese", dict[word]
```


Removing Dictionary Entries

- **Format**:
  - del <*dictionary_name*> [*key*]

- **Example**:
  del dict ["one"]

Example: Deletion And Checking For Membership

```
dict = {}
dict ["one"] = "Sentence one"
dict ["two"] = "Sentence two"
dict ["three"] = "Sentence three"

if "one" in dict:
   print "key one is in the dictionary"

del dict["one"]
if "one" not in dict:
   print "key one is NOT in the dictionary"
```

James Tam

You Should Now Know

- What is the difference between a mutable and an immutable type
- How strings are actually a composite type
- Common string functions and operations
- Why and when a list should be used
- How to create and initialize a list
- How to access or change the elements of a list
- Issues associated with copying lists and passing lists as parameters into functions
- When to use lists of different dimensions
- How to use the 'in' operator in conjunction with lists
- How a list can be used to store different types of information (non-homogeneous composite type)

James Tam

You Should Now Know (2)

- Common list operations and functions
- How to define an arbitrary composite type using a class
- What is a tuple and how do they differ from other composite types

James Tam

You Should Now Know (2)

- How to create a tuple and access the elements
- Why functions at most return a single value
- What is a dictionary and when can they can be used
- How to create a dictionary, access and remove elements

James Tam