

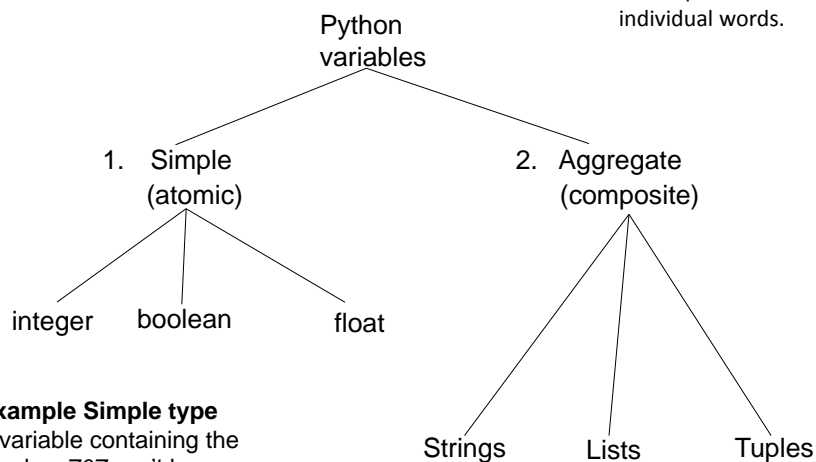
Composite Types

You will learn how to create new variables that are collections of other entities

Types Of Variables

Example composite

A string (collection of characters) can be decomposed into individual words.



Example Simple type

A variable containing the number 707 can't be meaningfully decomposed into parts

Small Example Programs Using Strings

- Basic string operations/concepts (some may have already been covered)
 - `String1.py` (strings as sets test for inclusion using 'in')
 - `String2.py` (iterating strings using the 'in' operator)
 - `String3.py` (Concatenation, repetition)
 - `String4.py`: (passing a whole string to a function)
 - `String5.py` (indexing the parts of a string)
 - `String6.py` (demonstrating the immutability of strings)
 - `String7.py` (converting to/from a string)

Small Example Programs Using Strings (2)

- New/more advanced string examples
 - `String8.py` (string slicing)
 - `String9.py` (string splitting)
 - `String10.py` (determining the size of strings)
 - `String11.py` (testing if strings meet certain conditions)
 - `String12.py` (ASCII values of characters)
- All the examples will be located in UNIX under:
`/home/courses/217/examples/composites`
- Also they can be found by looking at the course website under the URL:
 - <http://pages.cpsc.ucalgary.ca/~tamj/217/examples/composites>

Basic String Operations / Functions

- Some of these may have already been covered earlier during the semester

James Tam

Strings Can Be Conceptualized As Sets

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

```
User name: username
User name already taken, enter a new one
```

- Branching (example name: "string1.py")

```
userNames = "aaa abc username xxx"
```

```
userName = input("User name: ")
```

```
if userName in userNames:
```

```
    print("User name already taken, enter a new one")
```

```
User name: xxx
User name already taken, enter a new one
```

```
User name: charlie_sheen
```

- Looping (iterating through the elements: example name "string2.py"¹)

```
sentence = "by ur command"
```

```
for temp in sentence:
```

```
    print("%s-" %temp, end="")
```

```
b-y- -u-r- -c-o-m-m-a-n-d-
```

¹ Use of the write function requires the 'import' of the library sys: allows for more precise formatting than the standard print

String Operations: Concatenation & Repetition

- Concatenation ('+'): connects two or more strings
- Repetition (*): repeat a series of characters
- Complete online example: `string3.py`

```
s1 = "11"
s2 = "17"
s3 = s1 + s2
s4 = s2 * 3
print(s3) 1117
print(s4) 171717
```

James Tam

String: Composite

- Strings are just a series of characters (e.g., alpha, numeric, punctuation etc.)

- A string can be treated as one entity.
- Online example: “`string4.py`”

```
def fun(aString):
    print(aString)    By your command
```

```
# START
aString = "By your command"
fun(aString)
```

- Individual elements (characters) can be accessed via an index.

- Online example: “`string5.py`”
- Note: A string with ‘n’ elements has an index from 0 to (n-1)

```
aString = "hello"
print (aString[1])
print (aString[4])
```

```
[csl composites 63 ]> python string5.py
e
o
```

Mutable, Constant, Immutable,

- **Mutable types:**
 - The original memory location *can* change
- **Constants**
 - Memory location *shouldn't* change (Python): may produce a logic error if modified
 - Memory location syntactically *cannot* change (C++, Java): produces a syntax error (violates the syntax or rule that constants cannot change)
- **Immutable types:**
 - The *original* memory location *won't* change
 - Changes to a variable of a pre-existing immutable type creates a new location in memory. There are now two locations.

```
num = 12
num = 17
num 17
```

```
immutable = 12
immutable = 17
immutable 12
immutable 17
```

James Tam

Strings Are Immutable

- Even though it may look a string can change they actually cannot be edited (original memory location cannot change).
 - Online example: "string6.py"

```
s1 = "hi"
print (s1)
s1 = "bye"    # New string created
print (s1)
s1[0] = "G"  # Error
```

Converting To Strings

- Online example: `string7.py`

```
a = 2
b = 2.5
c = a + b # Addition
print(c) # Yields 4.5
```

```
# str() Converts argument to a String
# Convert to string and then concatenate
c = str(a) + str(b)
print(c) # Yields '22.5'
```

James Tam

Converting From Strings

```
x = '3'
y = '4.5'
# int(): convert to integer
# float(): convert to floating point
# Convert to numeric and then add
z = int(x) + float(y)
print(z) # Yields 7.5
```

James Tam

Advanced Operations / Functions

- These operations and functions likely have not yet been covered

James Tam

Substring Operations

- Sometimes you may wish to extract out a portion of a string.
 - E.g., Extract first name “James” from a full name “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*]

- **Online example:** string8.py

```
aString = "abcdefghij"
print (aString)      abcdefghij
temp = aString [2:5]
print (temp)         cde
temp = aString [:5]
print (temp)         abcde
temp = aString [7:]
print (temp)         hij
```

Example Use: String Slicing

- Where characters at fixed positions must be extracted.
- Example: area code portion of a telephone number
"403-210-9455"
 - The "403" area code could then be passed to a data base lookup to determine the province.

String Splitting

- Divide a string into portions with a particular character determining where the split occurs.
- Practical usage
 - The string “The cat in the hat” could be split into individual words (split occurs when spaces are encountered).
 - “The” “cat” “in” “the” “hat”
 - Each word could then be individually passed to a spell checker.

String Splitting (2)

- **Format:**
`string_name.split ('<character used in the split')`
- **Online example:** string9.py


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

James Tam

Determining Size

- The 'len()' function can count the number of characters in a string.
- Example program: string10.py

```
MAX_FILE_LENGTH = 256
SUFFIX_LENGTH = 3
```

```
Enter new file name (max 256 characters): a.txt
Text file
2:5 txt
```

```
filename = input("Enter new file name (max 256 characters): ")
if (len(filename) > MAX_FILE_LENGTH):
    print("File name exceeded the max size of %d characters, you bad"
          % (MAX_FILE_LENGTH))
else:
    # Find file type, last three characters in string e.g., resume.txt
    endSuffix = len(filename)
    startSuffix = endSuffix - SUFFIX_LENGTH
    suffix = filename[startSuffix:endSuffix]
    if (suffix == "txt"):
        print("Text file")
        print("%d:%d %s" % (startSuffix, endSuffix, suffix))
```

James Tam

String Testing Functions¹

- 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).

String Testing Functions (2)

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

Applying A String Testing Function

Name of the online example: "string11.py"

```
ok = False
while (ok == False):
    temp = input ("Enter an integer: ")
    ok = temp.isdigit()
    if (ok == False):
        print(temp, "is not an integer")
num = int (temp)
num = num + num
print(num)
```

```
Enter an integer: abc
abc is not an integer
```

```
Enter an integer: 11.2
11.2 is not an integer
```

```
Enter an integer: 12
24
```

Heuristic (end of "loops") applied also (good error message)

ASCII Values

- Each character is assigned an ASCII code e.g., 'A' = 65, 'b' = 98
- The `chr()` function can be used to determine the character (string of length one) for a particular ASCII code.
- The `ord()` function can be used to determine the ASCII code for a character (string of length one).

- Example: `string12.py`

```
aChar = input("Enter a character whose ASCII value that you wish to
see: ")
print("ASCII value of %s is %d" %(aChar,ord(aChar)))
```

```
aCode = int(input("Enter an ASCII code to convert to a character: "))
print("The character for ASCII code %d is %s" %(aCode,chr(aCode)))
```

```
Enter a character whose ASCII value that you wish to see: A
ASCII value of A is 65
```

```
Enter an ASCII code to convert to a character: 66
The character for ASCII code 66 is B
```

List

- In many programming languages a list is implemented as an array.
 - This will likely be the term to look for if you are looking for a list-equivalent when learning a new language.
- Python lists have many of the characteristics of the arrays in other programming languages but they also have other features.

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.

Why Bother With A List?

- Name of the example program: classList1.py

```
CLASS_SIZE = 5
```

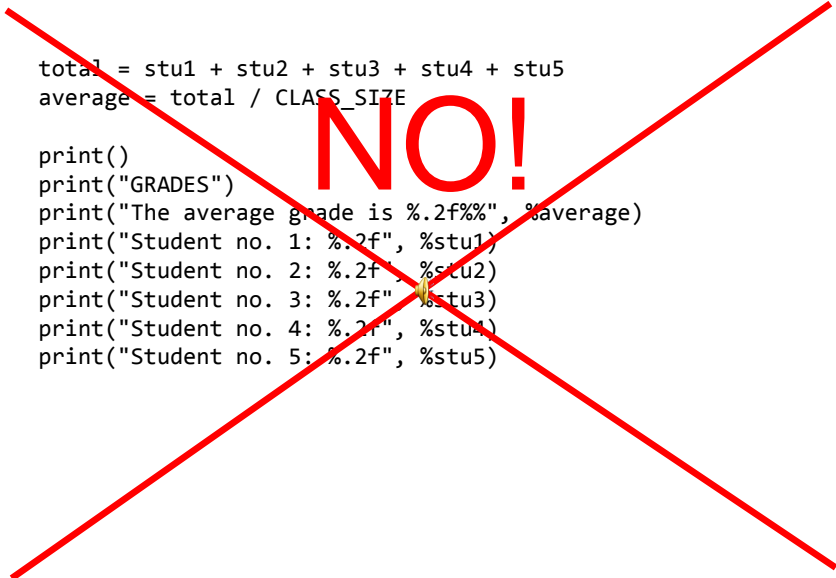
```
stu1 = float(input("Enter grade for student no. 1: "))  
stu2 = float(input("Enter grade for student no. 2: "))  
stu3 = float(input("Enter grade for student no. 3: "))  
stu4 = float(input("Enter grade for student no. 4: "))  
stu5 = float(input("Enter grade for student no. 5: "))
```

Why Bother With A List? (2)

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

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

Why Bother With A List? (3)



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

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

NO!

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.

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...a list!

Creating A List (Fixed Size)

•Format ('n' element list):

```
<list_name> = [Element 0<value 1>, Element 1<value 2>, ... Element n-1<value n>]
```

Example:

```
# List with 5 elements
```

```
percentages = [050.0, 1100.0, 278.5, 399.9, 465.1]
```

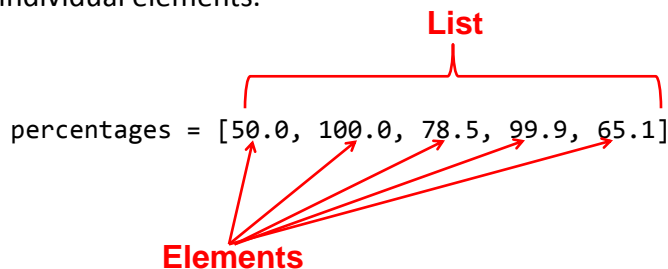
Other Examples:

```
letters = ['A', 'B', 'A']
```

```
names = ["The Borg", "Klingon ", "Hirogin", "Jem'hadar"]
```

Accessing A List

- Because a list is composite you can access the entire list or individual elements.



- Name of the list accesses the whole list

```
print(percentages)
```

```
>>> print(percentages)
[50.0, 100.0, 78.5, 99.9, 65.1]
```

- Name of the list and an index "[index]" accesses an element

```
print(percentages[1])
```

```
>>> print(percentages[1])
100.0
```

James Tam

Negative Indices

- Although Python allows for negative indices (-1 last element, -2 second last...-<size>) this is unusual and this approach is not allowed in other languages.
- So unless otherwise told your index should be a positive integer ranging from <zero> to <list size – 1>

James Tam

Creating A List (Variable Size)

- Step 1: Create a variable that refers to the list (more references later).

- **Format:**

```
<list name> = []
```

- **Example:**

```
classGrades = []
```

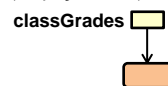
Creating A List (Variable Size: 2)

- Step 2: Initialize the list with the elements
- **General format:**
 - Within the body of a loop create each element and then add the new element on the end of the list ('append')

Creating A Variable Sized List: Example

```
classGrades = []
```

Before loop
(empty list)

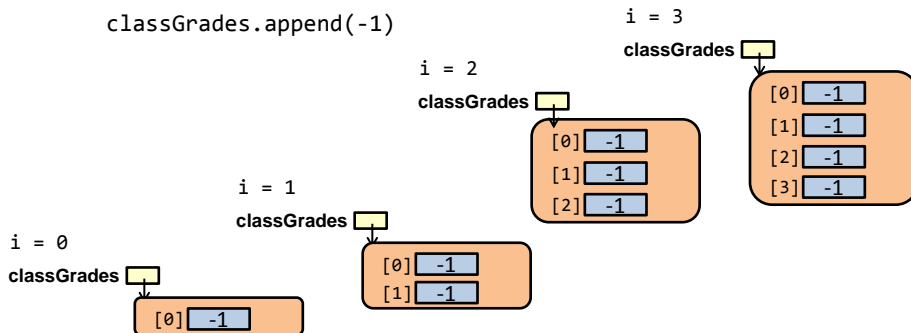


```
for i in range (0, 4, 1):
```

```
    # Each time through the loop: create new element = -1
```

```
    # Add new element to the end of the list
```

```
    classGrades.append(-1)
```



James Tam

Revised Version Using A List

•Name of the example program: classList2.py

```
CLASS_SIZE = 5
```

```
def initialize():
    classGrades = []
    for i in range (0, CLASS_SIZE, 1):
        classGrades.append(-1)
    return(classGrades)
```

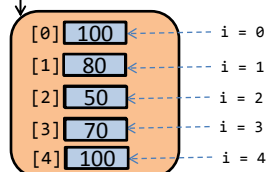
Revised Version Using A List (2)

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

```
Enter grade for student no. 1 :
>100
Enter grade for student no. 2 :
>80
Enter grade for student no. 3 :
>50
Enter grade for student no. 4 :
>70
Enter grade for student no. 5 :
>100
```

After 'initialize': before loop

classGrades



temp	1	2	3	4	5
Current grade	100	80	50	70	100
total	100	180	230	300	400
Loop ends now (Recall: CLASS_SIZE = 5)					
average	80	80			

Revised Version Using A List (3)

```
def display(classGrades, average):
    print()
    print("GRADES")
    print("The average grade is %.2f%%" %average)
    for i in range (0, CLASS_SIZE, 1):
        temp = i + 1
        print("Student No. %d: %.2f%%"
              %(temp,classGrades[i]))
```

```
GRADES
The average grade is 80.00%
Student No. 1: 100.00%
Student No. 2: 80.00%
Student No. 3: 50.00%
Student No. 4: 70.00%
Student No. 5: 100.00%
```

James Tam

Revised Version Using A List (4)

```
def start():
    classGrades = initialize()
    classGrades, average = read(classGrades)
    display(classGrades,average)

start()
```

James Tam

One Part Of The Previous Example Was Actually Unneeded

```
def read(classGrades):
    :           :
    return (classGrades, average)
```

When list is passed as a parameter

Returning the list is likely not needed

More details on 'why' coming up shortly!

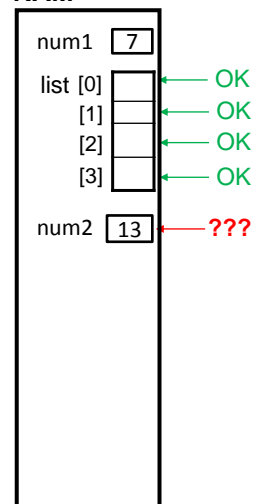
Take Care Not To Exceed The Bounds Of The List

Example: listBounds.py

```
num1 = 7
list = [0, 1, 2, 3]
num2 = 13
for i in range (0, 4, 1):
    print (list [i])

print ()
print (list [4])
```

RAM



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] = int(input ("Enter a value:" ))
```

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

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] = int(input ("Enter a value:" ))
```

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

Lists: Searching/Modifying, Len() Function

- Common problem: searching for matches that meet a certain criteria.
- The matches may simply be viewed or they may modified with a new value.
- Example: listFindModify.py

```
grades = ['A','B','C','A','B','C','A']
last = len(grades) - 1
i = 0
while (i <= last):
    if (grades[i] == 'A'): # Search for matches
        grades[i] = 'A+' # Modify element
    i = i + 1
print(grades)
```

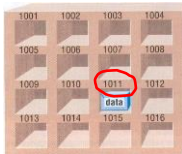
```
['A+', 'B', 'C', 'A+', 'B', 'C', 'A+']
```

James Tam

Recap: Variables

- Variables are a 'slot' in memory that contains 'one piece' of information.

```
num = 123
```



Picture from Computers in your future by Pfaffenberger B

- Normally a location is accessed via the name of the variable.
 - Note however that each location is also numbered!

James Tam

Recap: Assignment (Simple Types)

```
num1 = 2
num2 = 3
num1 = num2
```

Copy contents from
memory location called
'num2' into location called
'num1'

James Tam

List Variables Are References To Lists (Not Actual Lists)

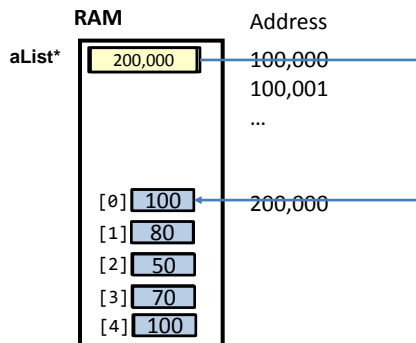
- Most of the time the difference between a reference to a list and the actual list is not noticeable.
- However there will be times that it's important to make that distinction e.g., using the assignment operator, passing parameters.
- Small example:

```
aList = []
aList = [1,2,3]
```

Create list
Put address in
reference

Note

- A reference to a list actually contains an address.
- An 'empty list' contains no address yet
- A non-empty list contains the address of the list



James Tam

Example: List References

```
list1 = [1,2,3]
list2 = list1
# Looks like two lists, actually just two references to one list
```

```
print(list1,list2) [1, 2, 3] [1, 2, 3]
```

```
list1 = [3,2,1]
# List1 refers to a new list
print(list1,list2)
```

```
[3, 2, 1] [1, 2, 3]
```

James Tam

Copying Lists

- If you use the assignment operator to copy from list to another you will end up with only one list).
- Name of the example program: copyList1.py

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

```
[1, 2] [2, 1]
```

```
# Two ref to one list
```

```
list1 = list2
print (list1, list2)
```

```
[2, 1] [2, 1]
```

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

```
[99, 1] [99, 1]
```

Copying Lists (2)

- To copy the elements of one list to another a loop is needed to copy each element.
- Name of the example program: copyList2.py

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

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

print(list1, list2) [1, 2, 3] [1, 2, 3]
list1[1] = 99
print(list1, list2) [1, 99, 3] [1, 2, 3]
```

Recap: Parameter Passing (Simple Types)

```
def fun(num):
    print(num) 1
    num = num + num
    print(num) 2

def start():
    num = 1
    print(num) 1
    fun(num)
    print(num) 1

start()
```

James Tam

Passing Lists As Parameters

- When a list variable is passed into function it's not actually the whole list that is passed.

```
def start():
    aList = [1,2,3]
    fun(aList)
```

Address of list passed
(memory efficient if list is
large and function is
called many times)

- Instead it's just a reference to a list (address) that is passed into the function (which then stores the address in a local variable)

```
def fun(aList):
```

The address is stored in a
local variable

James Tam

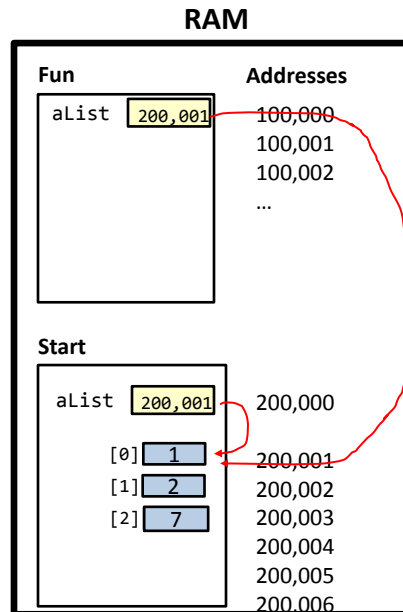
Passing Lists As Parameters

```
def fun(aList):
    aList[2] = 7
```

```
def start():
    aList = []

    aList = [1,2,3]

    fun(aList)
```



James Tam

Example: Passing Lists As Parameters

- Name of complete example: listParameters.py

```
def fun1(aListCopy):
    aListCopy[0] = aListCopy[0] * 2
    aListCopy[1] = aListCopy[1] * 2
    return aListCopy

def fun2(aListCopy):
    aListCopy[0] = aListCopy[0] * 2
    aListCopy[1] = aListCopy[1] * 2
```

James Tam

Example: Passing Lists As Parameters (2)

```
def start():
    Original list in start() before function calls: [2, 4]
    alist = [2,4]
    print("Original list in start() before function
          calls:\t", end="")
    print(alist)
    alist = fun1(alist)
    print("Original list in start() after calling fun1():\t",
          end="")
    Original list in start() after calling fun1(): [4, 8]
    print(alist)
    fun2(alist)
    print("Original list in start() after calling fun2():\t",
          end="")
    print(alist)
    Original list in start() after calling fun2(): [8, 16]
start()
```

James Tam

Why Are References Used?

- It looks complex
- Most important reason why it's done: efficiency
 - Since a reference to a list contains the address of the list it allows access to the list.
 - As mentioned if the list is large and a function is called many times the allocation (creation) and de-allocation (destruction/freeing up memory for the list) can reduce program efficiency.
- Size of references ~range 32 bits (4 bytes) to 64 bits (8 bytes)
- Contrast this with the size of a list
 - E.g., a list that refers to online user accounts (each account is a list element that may be multi-Giga bytes in size). Contrast passing an 8 byte reference to the list vs. passing a multi-Gigabyte list.

James Tam

Simulation, What If A List And Not A List Reference Passed: Creating A New List Each Function Call

- Name of full online example: `listExampleSlow.py`

```

MAX = 1000000

def fun(i):
    print("Number of times function has been called %d" %(i))
    aList = []
    for j in range (0,MAX,1):
        aList.append(str(j))

def start():
    for i in range (0,MAX,1):
        fun(i)

start()

```

James Tam

Formatting Output: Newline

- # Method 1: Review (any variable type, displays all on one line)

```
print("hello", end = "")
```

- # Method 2: New (a string variable only – displays all on one line)

```
import sys
sys.stdout.write("hello")
```

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:

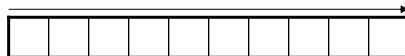
- (1D list)

–Tracking grades for a class (previous example)

–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 list)

–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

When To Use Lists Of Different Dimensions (2)

- (2D list continued)

Lecture section	Student			
	First student	Second student	Third student	...
L01				
L02				
L03				
L04				
L05				
:				
L0N				

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]

Specifying only a single value specifies the row

	Columns			
	[0]	[1]	[2]	[3]
[0]	L01			
[1]	L02			
[2]	L03			
[3]	L04			
[4]	L05			
[5]	L06			
[6]	L07			

Rows

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

General structure

```
<list_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): Fixed Size

Name of the example program: display2DList.py

```
matrix = [ [0, 0, 0],
            [1, 1, 1],
            [2, 2, 2],
            [3, 3, 3]]
i = 0
i = 1
i = 2
i = 3
for r in range (0, 4, 1):
    print (matrix[r]) # Each print displays a 1D list
```

```
for r in range (0,4, 1):
    for c in range (0,3,1):
        sys.stdout.write(str(matrix[r][c]))
    print()
```

```
print(matrix[2][0]) #2 not 0
```

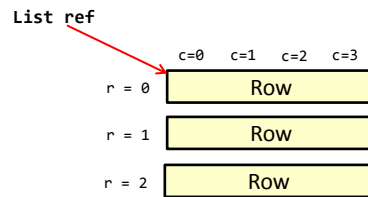
```
000
111
222
333
```

```
2
```


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

General structure (Using loops):

- Create a variable that refers to a 1D list.
- One loop (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.
- Repeat the process for each row in the list



Etc.

Creating And Initializing A Multi-Dimensional List In Python (4)

• 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 an empty 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
                                # (space) of the 1D list
```

James Tam

Example 2D List Program: A Character-Based Grid

- **Name of the example program:** `simple_grid.py`

```
import sys

aGrid = []

for r in range (0,2,1):
    aGrid.append ([])
    for c in range (0,3,1):
        aGrid[r].append (str(r+c))

for r in range (0,2,1):
    for c in range (0,3,1):
        sys.stdout.write(str(aGrid[r][c]))
    print()
```

Quick Note” List Elements Need Not Store The Same Data Type

- This is one of the differences between Python lists and arrays in other languages
- Example:
`alist = ["James", "Tam", "210-9455", 707]`

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.

Creating Tuples

- **Format:**
`tuple_name = (value1, value2...valuen)`
- **Example:**
`tup = (1,2,"foo",0.3)`

A Small Example Using Tuples

- Name of the online example: tuples1.py

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

```
(1, 2, 'foo', 0.3)
foo
```

Error (immutable):
 "TypeError: object does not support item assignment"

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

Functions Changing Multiple Items

- Because functions only return 0 or 1 items (Python returns one composite) the mechanism of passing by reference (covered earlier in this section) is an important concept.
 - What if more than one change must be communicated back to the caller (only one entity can be returned).
 - Multiple parameters can be passed by reference.

Extra Practice

String:

- Write the code that implements string operations (e.g., splitting) or string functions (e.g., determining if a string consists only of numbers)

List operations:

- For a numerical list: implement some common mathematical functions (e.g., average, min, max, mode).
- For any type of list: implement common list operations (e.g., displaying all elements one at a time, inserting elements at the end of the list, insert elements in order, searching for elements, removing an element).

After This Section You Should Now Know

- The difference between a simple vs. a composite type
- 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 (fixed and dynamic size)
- How to access or change the elements of a list
- How to search a list for matches
- Copying lists: How does it work/How to do it properly

After This Section You Should Now Know (2)

- When to use lists of different dimensions
- Basic operations on a 2D list
- What is a tuple, common operations on tuples such as creation, accessing elements, displaying a tuple or elements
- How functions return zero or one item
- What is a reference and how it differs from a regular variable
- Why references are used
- The two parameter passing mechanisms: pass-by-value and pass-by-reference
- How to use the write() function in the 'system' library