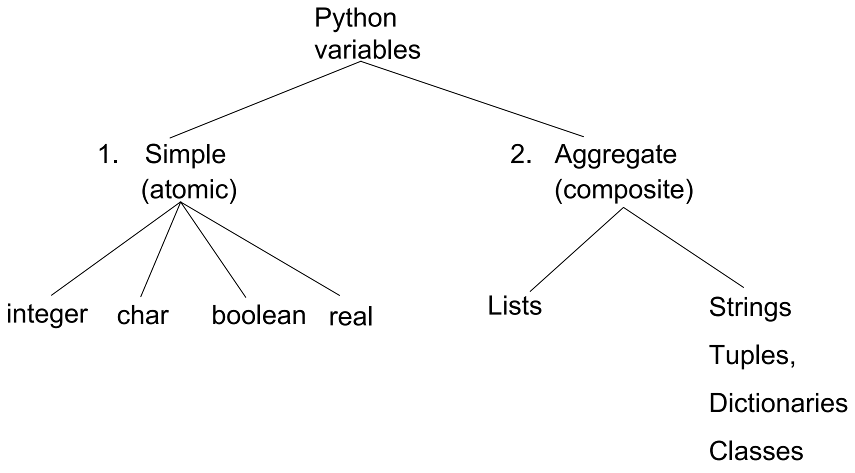
Lists

In this section of notes you will be introduced to new type of variable that consists of other types.

Types Of Variables



Python variables

1. Simple (atomic)

integer char boolean real

2. Aggregate (composite)

Lists

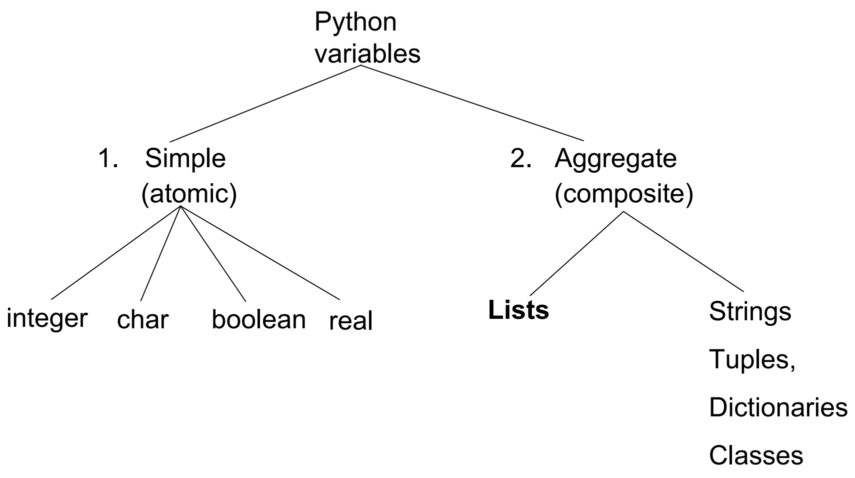
Strings

Tuples,

Dictionaries

Classes

Types Of Variables



Python variables

1. Simple (atomic)

integer

char

boolean

real

2. Aggregate (composite)

Lists

Strings

Tuples,

Dictionaries

Classes

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 section will talk about the features of lists that are largely common to arrays.

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 Composite Types?

- For the full version of the example look in UNIX under: /home/231/examples/lists/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: ")
```

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
```

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
```

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…an array / list!

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:**

  <array 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)
```

Revised Version Using A List

- For a full example look in UNIX under:

/home/231/examples/lists/classList2.py

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

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

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
```

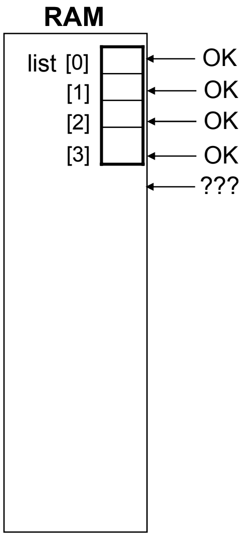
Output:

```
[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]←——???
```



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]
```

One Way Of Avoiding An Overflow Of The List

- Use a constant in conjunction with the list.

```
SIZE = 100000
```

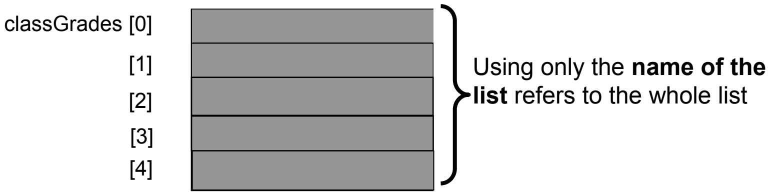
- The value in the constant controls traversals of the list

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

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

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”



classGrades [0]

[1]

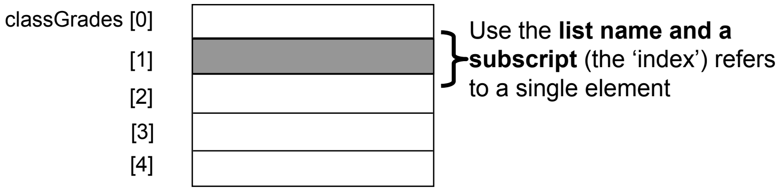
[2]

[3]

[4]

Using only the **name of the list** refers to the whole list

- 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]”



classGrades [0]

[1]

[2]

[3]

[4]

Use the **list name and a subscript** (the ‘index’) refers to a single element

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

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 version can be found in UNIX under: /home/231/examples/lists/copy1.py

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

list1 = list2
print list1, list2

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

Copying Lists (2)

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

Parameter Passing

- What you’ve seen so far:
  - Passing a parameter into a function makes a local copy of the value passed in.
  - This is referred to as **PASS BY VALUE**.
  - Changes made to the parameter will only be made to the local copy and not the original.

Parameter Passing (2)

- 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 version can be found in UNIX under: /home/231/examples/lists/parameter1.py

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


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

main ()
```

  - Changes made to the local reference will change the original list.
  - This parameter passing mechanism is referred to as **PASS BY REFERENCE** (the local reference refers to the original list)

Parameter Passing (3)

- Exception: 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 version of the program can be found in UNIX under: /home/231/examples/lists/parameter2.py

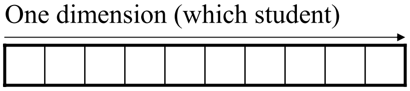
```
def fun (list):
   list = [3,2,1]
   print list

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

main ()
```

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

When To Use Lists Of Different Dimensions (2)

- (2D list continued)

Student

Lecture section

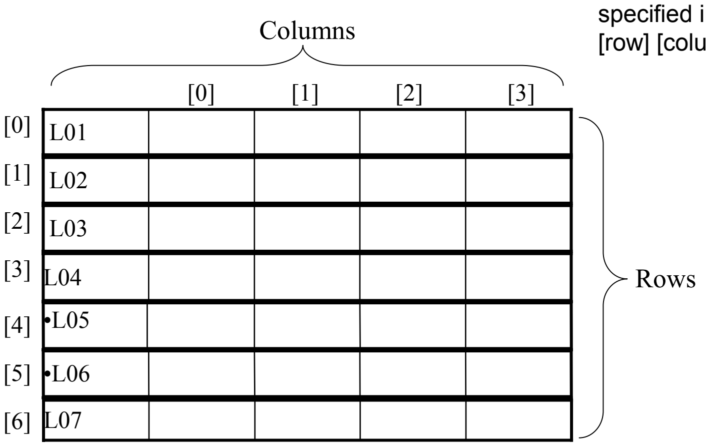
	First student	Second student	Third student	…
L01				
L02				
L03				
L04				
L05				
:				
L0N				

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]



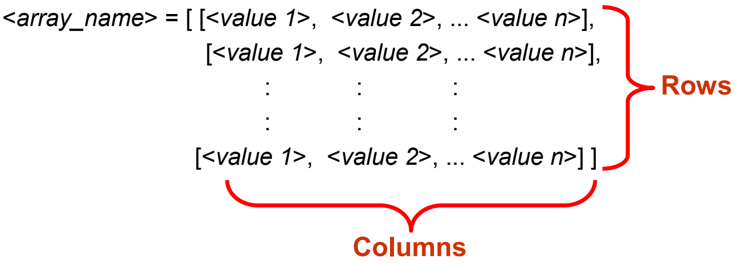
Columns

		[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

General structure



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

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

- You can find the full program in UNIX under:
/home/231/examples/lists/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)
```

A Character-Based Grid (4)

```
def display (aGrid):
    for r in range (1, MAX_ROWS, 1):
        for c in range (1, 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 " - - - -"
```

A Character-Based Grid (5)

- **# MAIN FUNCTION**

```
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)
```

Lists Can Be Treated As A Set

- That means that the ‘in’ operator can be used in conjunction with lists.
- Branching

```
list = ["bob", "alice", "mary", "tom", "dick", "harry"]
if ("tom" in list):
    print "tom is in"
```

- Loops

```
list = [123, 43, 35, 1, 888, 666, 777]
temp = 0
for temp in list:
    print temp
```

You Should Now Know

- 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

After This Section You Should Now Know

- How to write the definition for a function
  - How to write a function call
- How to pass information to and from functions via parameters and return values
- How and why to declare variables locally
- How to test functions and procedures
- How to design a program from a problem statement
  - How to determine what are the candidate functions
  - How to determine what variables are needed and where they need to be declared
  - Some approaches for developing simple algorithms (problem solving techniques)