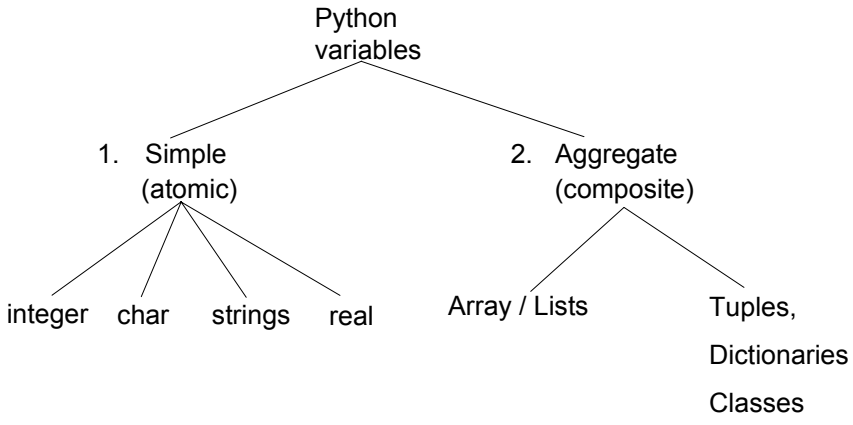
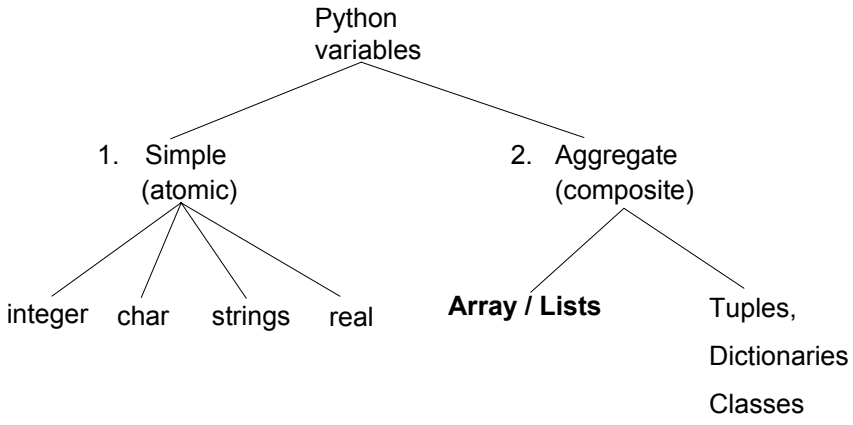
Arrays / Lists

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

Types Of Variables



Types Of Variables



Arrays / Lists

Arrays

- In most programming languages this composite type is referred to as an array.

Lists

- The Python implementation of an array is a list.
- A list can be used and manipulated as an array (focus of this section).
- However the Python list is more powerful than a regular array (to be discussed in a later section).

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?

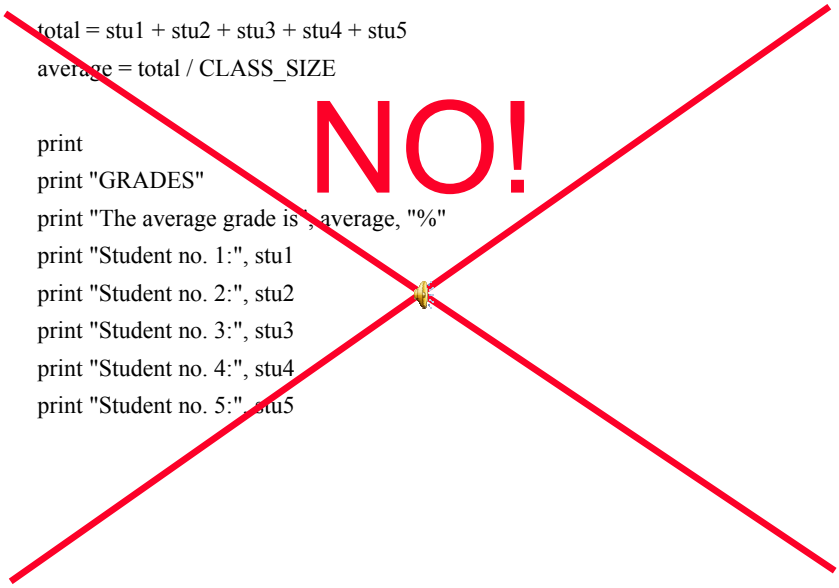
For a compilable example look in UNIX under:
/home/courses/217/examples/arrays/classList1.py

```
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
```

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 An Array (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.

General structure (Fixed sized array):

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

Example (Fixed sized array):

percentages = [0.0, 0.0, 0.0, 0.0, 0.0]

letters = ['A', 'A', 'A']

names = ["James Tam", "Stacey Walls", "Jamie Smyth"]

Creating An Array (With Loops)

Step 1: Create a variable that refers to an array

Format:

<array name> = []

Example:

classGrades = []

Creating An Array (With Loops: 2)

Step 2: Initialize the array with the array elements

General format:

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

Example:

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

Revised Version Using An Array

For a full example look in UNIX under:
/home/courses/217/tamj/examples/arrays/classList2.py

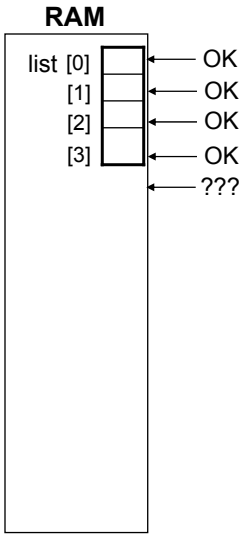
```
CLASS_SIZE = 5

# Read: Step through the array an element at a time and get the user to input the
# grades.
def read (classGrades, average):
    total = 0

    for i in range (0, CLASS_SIZE, 1):
        # Because array indices start at zero add one to the student number.
        temp = i + 1
        print "Enter grade for student no.", temp, ":",
        classGrades[i] = input ()
        total = total + classGrades[i]
        average = total / CLASS_SIZE
    return (classGrades, average)
```


Revised Version Using An Array (2)

```
# Display: Traverse the array and display the grades.
def display (classGrades, average):
    print
    print "GRADES"
    print "The average grade is", average, "%"
    for i in range (0, CLASS_SIZE, 1):
        # Because array indices start at zero add one to the student number.
        temp = i + 1
        print "Student no.", temp, ":", classGrades[i], "%"

# MAIN FUNCTION
i = 0
temp = 0
average = 0
classGrades = []
for i in range (0, CLASS_SIZE, 1):
        classGrades.append(0)

classGrades, average = read (classGrades, average)
display (classGrades, average)
```

Take Care Not To Exceed The Bounds Of Arrays

```
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 An Array

Use a constant in conjunction with arrays.

```
SIZE = 100
```

The value in the constant determines the size of the array (this is an alternative way to create and initialize an array).

```
myArray = ["^-^" for i in range (0, SIZE, 1)]
```

The value in the constant controls traversals of the array

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

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

One Way Of Avoiding An Overflow Of An Array

Use a constant in conjunction with arrays.

SIZE = 100000

The value in the constant determines the size of the array (this is an alternative way to create and initialize an array).

```
myArray = ["^-^" for i in range (0, SIZE, 1)]
```

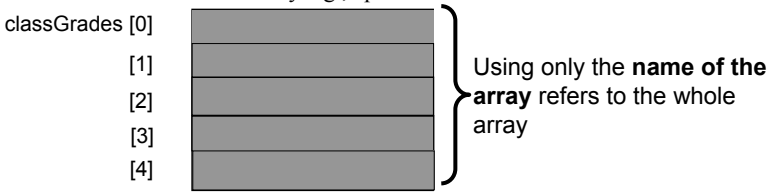
The value in the constant controls traversals of the array

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

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


Accessing Data In The Array

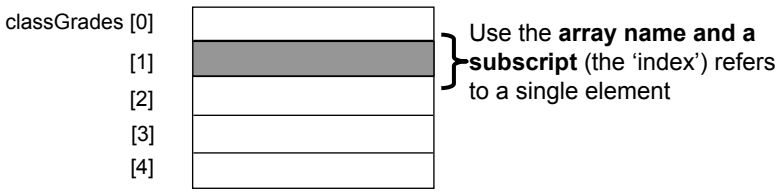
To manipulate an array you need to first indicate which array is being accessed

- Done via the name of the array 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 array index e.g., "print classGrades[1]"

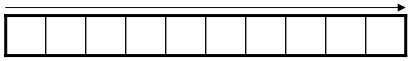


When To Use Arrays Of Different Dimensions

- 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 Arrays Of Different Dimensions (2)

- (2D array continued)

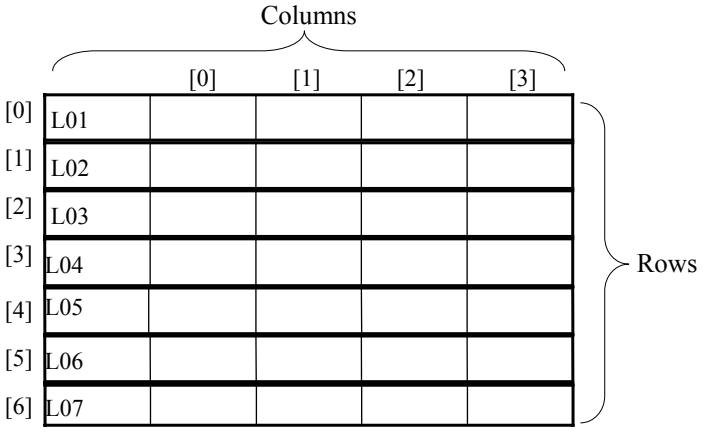
Student

Lecture section

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

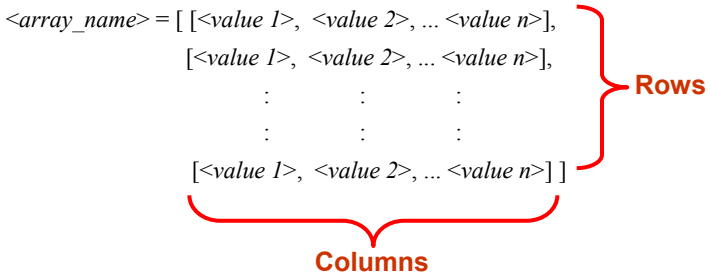
When To Use Arrays Of Different Dimensions (3)

- (2D array continued)
- Notice that each row is merely a 1D array
- (A 2D array is an array containing rows of 1D arrays)



Creating And Initializing A Multi-Dimensional Array In Python

General structure (Fixed sized array):



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

Example (Fixed sized array):

```
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 Array In Python (3)

General structure (Variable sized array):

Create a variable that refers to a 1D array. The outer loop traverses the rows. For each iteration of the outer loop create a new 1D array. Then with the inner loop traverse the columns of the newly created 1D array creating and initializing each element in a fashion similar to how a single 1D array was created and initialized.

Example (Variable sized array):

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

Example 2D Array Program: A Character-Based Grid

You can find the full program in Unix under:
/home/courses/217/examples/arrays/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)
    return aGrid
```


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 (" ")

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

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


You Should Now Know

- Why and when should an array be used
- How to create and initialize an array
- How to access or change the elements
- When to use arrays of different dimensions