COMP 10280
Programming I (Conversion)

John Dunnion

School of Computer Science
University College Dublin

COMP 10280 Programming I (Conversion)/Lecture 16

Outline

Python program to count numbers (1)

```
# Program to count numbers using variables

# Initialise all counters to 0
count_0 = 0
count_1 = 0
count_2 = 0
count_3 = 0

# Prompt the user for a number
number = int(input('Enter a number (an int >= 0 and <= 3):  '))
while 0 <= number <= 3:
    if number == 0:
        count_0 += 1
    elif number == 1:
        count_1 += 1
    elif number == 2:
        count_2 += 1
    elif number == 3:
        count_3 += 1

# Prompt the user for another number
    number = int(input('Enter a number (an int >= 0):  '))
```

Python program to count numbers (2)

```
# Print the results
print('Number of 0:', count_0)
print('Number of 1:', count_1)
print('Number of 2:', count_2)
print('Number of 3:', count_3)

print('Finished!')
```

Python program to count numbers (3)

```
Enter a number (an int >= 0 and <= 3):  1
Enter a number (an int >= 0 and <= 3):  2
Enter a number (an int >= 0 and <= 3):  3
Enter a number (an int >= 0 and <= 3):  1
Enter a number (an int >= 0 and <= 3):  2
Enter a number (an int >= 0 and <= 3):  3
Enter a number (an int >= 0 and <= 3):  0
Enter a number (an int >= 0 and <= 3):  1
Enter a number (an int >= 0 and <= 3):  2
Enter a number (an int >= 0 and <= 3):  3
Enter a number (an int >= 0 and <= 3):  4
Number of 0:  1
Number of 1:  3
Number of 2:  3
Number of 3:  3
Finished!
```

Arrays

- We often have the need to carry out the same operations on a number of items
- Most programming languages provide a way of describing a collection of variables with identical properties
- This collection is called the array
- There is usually a single name for all of the members of the collection
- Individual members are selected using an index
- In C, `int x[10];` declares an array of ten locations, each of type `int`
- In Java, `int[] x;` declares an array of `int`s
- We can access an individual element of the array, for example `x[6]`

Arrays

- We often have the need to carry out the same operations on a number of items
- Most programming languages provide a way of describing a collection of variables with identical properties
- This collection is called the array
- There is usually a single name for all of the members of the collection
- Individual members are selected using an index
- In C, `int x[10];` declares an array of ten locations, each of type `int`
- In Java, `int[] x;` declares an array of `int`s
- We can access an individual element of the array, for example `x[6]`

Arrays

- We often have the need to carry out the same operations on a number of items
- Most programming languages provide a way of describing a collection of variables with identical properties
- This collection is called the array
- There is usually a single name for all of the members of the collection
- Individual members are selected using an index
- In C, `int x[10];` declares an array of ten locations, each of type `int`
- In Java, `int[] x;` declares an array of `int`s
- We can access an individual element of the array, for example `x[6]`

Arrays

- We often have the need to carry out the same operations on a number of items
- Most programming languages provide a way of describing a collection of variables with identical properties
- This collection is called the array
- There is usually a single name for all of the members of the collection
- Individual members are selected using an index
- In C, `int x[10];` declares an array of ten locations, each of type `int`
- In Java, `int[] x;` declares an array of `int`s
- We can access an individual element of the array, for example `x[6]`

Arrays

- We often have the need to carry out the same operations on a number of items
- Most programming languages provide a way of describing a collection of variables with identical properties
- This collection is called the array
- There is usually a single name for all of the members of the collection
- Individual members are selected using an index
- In C, `int x[10];` declares an array of ten locations, each of type `int`
- In Java, `int[] x;` declares an array of `int`s
- We can access an individual element of the array, for example `x[6]`

Scalar types and structured types in Python

- We have seen a number of scalar types in Python
- The numeric types `int` and `float` are scalar types
- It is not possible to access their internal structure
- We have also seen a structured type or non-scalar type: the `str` type
- We can use indexing to extract individual characters and slicing to extract substrings

Scalar types and structured types in Python

- We have seen a number of scalar types in Python
- The numeric types `int` and `float` are scalar types
- It is not possible to access their internal structure
- We have also seen a structured type or non-scalar type: the `str` type
- We can use indexing to extract individual characters and slicing to extract substrings

Scalar types and structured types in Python

- We have seen a number of scalar types in Python
- The numeric types `int` and `float` are scalar types
- It is not possible to access their internal structure
- We have also seen a structured type or non-scalar type: the `str` type
- We can use indexing to extract individual characters and slicing to extract substrings

Scalar types and structured types in Python

- Python does not have arrays!
- Python does have a number of other structured types that provide for collections of elements

Scalar types and structured types in Python

- Python does not have arrays!
- Python does have a number of other structured types that provide for collections of elements

Tuples

- Like strings, a tuple in Python is an ordered sequence of elements
- Unlike strings, the elements of a tuple:
  - need not be characters
  - can be of any type
  - need not be of the same type as each other
- Literals of type `tuple` are written by enclosing a comma-separated list of elements within parentheses

Tuples

- Like strings, a tuple in Python is an ordered sequence of elements
- Unlike strings, the elements of a tuple:
  - need not be characters
  - can be of any type
  - need not be of the same type as each other
- Literals of type `tuple` are written by enclosing a comma-separated list of elements within parentheses

Using tuples

The program below:

```
# Program to demonstrate the use of tuples

t1 = (1, 2, 3)
t2 = (4, 'five', 6.50, 7)
t3 = ()

print('Printing the tuples:'
print('t1 is:', t1)
print('t2 is:', t2)
print('t3 is:', t3)

print('Finished!')
```

produces the following output:

```
Printing the tuples:
t1 is:  (1, 2, 3)
t2 is:  (4, 'five', 6.5, 7)
t3 is:  ()
Finished!
```

More on tuples

- The tuple containing a single value `1` is not written `(1)`
- `(1)` evaluates to 1!
- To denote the singleton tuple containing the value 1, we write `(1,)`
- Note the comma!
- Like strings, tuples can be
  - concatenated;
  - indexed; and
  - sliced
- The `for` statement can be used to iterate over the elements of a tuple

More on tuples

- The tuple containing a single value `1` is not written `(1)`
- `(1)` evaluates to 1!
- To denote the singleton tuple containing the value 1, we write `(1,)`
- Note the comma!
- Like strings, tuples can be
  - concatenated;
  - indexed; and
  - sliced
- The `for` statement can be used to iterate over the elements of a tuple

More on tuples

- The tuple containing a single value `1` is not written `(1)`
- `(1)` evaluates to 1!
- To denote the singleton tuple containing the value 1, we write `(1,)`
- Note the comma!
- Like strings, tuples can be
  - concatenated;
  - indexed; and
  - sliced
- The `for` statement can be used to iterate over the elements of a tuple

More on tuples

- The tuple containing a single value `1` is not written `(1)`
- `(1)` evaluates to 1!
- To denote the singleton tuple containing the value 1, we write `(1,)`
- Note the comma!
- Like strings, tuples can be
  - concatenated;
  - indexed; and
  - sliced
- The `for` statement can be used to iterate over the elements of a tuple

Operations on tuples

The program below:

```
# Program to demonstrate operations on tuples

t1 = (1, 'two', 3.0)
t2 = (t1, 98.765)

print('t1 is:', t1)
print('t2 is:', t2)
print('(t1 + t2) is:', (t1 + t2))
print('(t1 + t2)[2] is:', (t1 + t2)[2])
print('(t1 + t2)[3] is:', (t1 + t2)[3])
print('(t1 + t2)[2:5] is:', (t1 + t2)[2:5])

print('Finished!')
```

produces the following output:

```
t1 is:  (1, 'two', 3.0)
t2 is:  ((1, 'two', 3.0), 98.765)
(t1 + t2) is:  (1, 'two', 3.0, (1, 'two', 3.0), 98.765)
(t1 + t2)[2] is:  3.0
(t1 + t2)[3] is:  (1, 'two', 3.0)
(t1 + t2)[2:5] is:  (3.0, (1, 'two', 3.0), 98.765)
Finished!
```

Using the `for` statement on a tuple

The program below:

```
# Program to demonstrate the use of the for statement on a tuple

t1 = (1, 'two', 3.0)

print('t1 is:', t1)
print('The elements of t1 are:')
for x in t1:
    print(x)

print('Finished!')
```

produces the following output:

```
t1 is:  (1, 'two', 3.0)
The elements of t1 are:
1
two
3.0
Finished!
```

Finding common divisors (1)

- Write a program that prompts the user for two positive integers, finds and prints out their common divisors, sums the common divisors and prints out the total

Prompt the user for two positive integers
if the numbers entered are not positive **then**
    print out an error message
else
    find the common divisors
    print out the common divisors
    sum the common divisors
    print out the total

Finding common divisors (2)

```
function findDivisors(num1, num2)
initialise divisors
for i from 1 to minmum(num1, num2) do
      if num1 mod i == 0 and num2 mod i == 0 then
            add i to divisors
return divisors
```

Finding common divisors program (1)

```
# Program to get the common divisors of two positive integers supplied
# Demonstrates the use of tuples

def findDivisors(num1, num2):
    """Finds the common divisors of num1 and num2

    Assumes that num1 and num2 are positive integers
    Returns a tuple containing the common divisors of num1 and num2"""

    divisors = ()
    for i in range(1, min(num1, num2) + 1):
        if num1 % i == 0 and num2 % i == 0:
            divisors += (i,)

    return divisors
```

Finding common divisors program (2)

```
number1 = int(input('Enter a positive integer:  '))
number2 = int(input('Enter another positive integer:  '))

if number1 <= 0 or number2 <= 0:
    print('Numbers should be > 0.')
else:
# First of all, get the common divisors and print them out
    divisors = findDivisors(number1, number2)
    print('The common divisors of', number1, 'and', number2, 'are:', d

# Now sum them and print the total
    total = 0
    for d in divisors:
        total += d
    print('Sum of the common divisors is:', total)

print('Finished!')
```

Lists

- Like tuples, a list in Python is an ordered sequence of elements
- Like tuples, the elements of a list can be of any type and need not be of the same type as each other
- Like tuples, lists can be concatenated, indexed and sliced
- Like tuples, the `for` statement can be used to iterate over the elements of a list
- Literals of type `list` are written by enclosing a comma-separated list of elements within square brackets
- An empty list is written as `[]`
- The singleton list containing the value 1 is written as `[1]` (Note: no comma!)

Lists

- Like tuples, a list in Python is an ordered sequence of elements
- Like tuples, the elements of a list can be of any type and need not be of the same type as each other
- Like tuples, lists can be concatenated, indexed and sliced
- Like tuples, the `for` statement can be used to iterate over the elements of a list
- Literals of type `list` are written by enclosing a comma-separated list of elements within square brackets
- An empty list is written as `[]`
- The singleton list containing the value 1 is written as `[1]` (Note: no comma!)

Lists

- Like tuples, a list in Python is an ordered sequence of elements
- Like tuples, the elements of a list can be of any type and need not be of the same type as each other
- Like tuples, lists can be concatenated, indexed and sliced
- Like tuples, the `for` statement can be used to iterate over the elements of a list
- Literals of type `list` are written by enclosing a comma-separated list of elements within square brackets
- An empty list is written as `[]`
- The singleton list containing the value 1 is written as `[1]` (Note: no comma!)

Lists

- Like tuples, a list in Python is an ordered sequence of elements
- Like tuples, the elements of a list can be of any type and need not be of the same type as each other
- Like tuples, lists can be concatenated, indexed and sliced
- Like tuples, the `for` statement can be used to iterate over the elements of a list
- Literals of type `list` are written by enclosing a comma-separated list of elements within square brackets
- An empty list is written as `[]`
- The singleton list containing the value 1 is written as `[1]` (Note: no comma!)

Lists

- Like tuples, a list in Python is an ordered sequence of elements
- Like tuples, the elements of a list can be of any type and need not be of the same type as each other
- Like tuples, lists can be concatenated, indexed and sliced
- Like tuples, the `for` statement can be used to iterate over the elements of a list
- Literals of type `list` are written by enclosing a comma-separated list of elements within square brackets
- An empty list is written as `[]`
- The singleton list containing the value 1 is written as `[1]` (Note: no comma!)

Using lists

The program below:

```
 Program to demonstrate the use of lists

l1 = [1, 2, 3]
l2 = [4, 'five', 6.50, 7]
l3 = []
l4 = [100]
l5 = ['another single element']

print('Printing the lists:'
print('l1 is:', l1)
print('l2 is:', l2)
print('l3 is:', l3)
print('l4 is:', l4)
print('l5 is:', l5)
```

produces the following output:

```
Printing the lists:
l1 is:  [1, 2, 3]
l2 is:  [4, 'five', 6.5, 7]
l3 is:  []
l4 is:  [100]
l5 is:  ['another single element']
```

Operations on lists

The program below:

```
# Program to demonstrate operations on lists

l1 = [1, 'two', 3.0]
l2 = [l1, 98.765]

print('l1 is:', l1)
print('l2 is:', l2)
l3 = l1 + l2
print('l3 (= l1 + l2) is:', l3)
print('l3[2] is:', l3[2])
print('l3[3] is:', l3[3])
print('l3[2:5] is:', l3[2:5])

print('Finished!')
```

produces the following output:

```
l1 is:  [1, 'two', 3.0]
l2 is:  [[1, 'two', 3.0], 98.765]
l3 (= l1 + l2) is:  [1, 'two', 3.0, [1, 'two', 3.0], 98.765]
l3[2] is:  3.0
l3[3] is:  [1, 'two', 3.0]
l3[2:5] is:  [3.0, [1, 'two', 3.0], 98.765]
Finished!
```

Using the `for` statement on a list

The program below:

```
# Program to demonstrate the use of the for statement on a list

l1 = [1, 'two', 3.0]

print('l1 is:', l1)
print('The elements of l1 are:')
for x in l1:
    print(x)

print('Finished!')
```

produces the following output:

```
l1 is:  [1, 'two', 3.0]
The elements of l1 are:
1
two
3.0
Finished!
```

Using the `for` statement on a list

The program below:

```
# Program to demonstrate the use of an index into a list

l1 = [1, 'two', 3.0]

print('The elements of l1 are:')
for i in range(len(l1)):
    print(l1[i])

print('Finished!')
```

produces the following output:

```
The elements of l1 are:
1
two
3.0
Finished!
```

Lists and tuples: mutable and immutable

- Lists differ from tuples and strings in one very important respect
- Lists are mutable
- Tuples and strings are immutable
- There are many operators that can be used to create objects of these immutable types, and variables can be bound to objects of these types
- However, objects of immutable types cannot be modified
- Objects of mutable types can be modified

Lists and tuples: mutable and immutable

- Lists differ from tuples and strings in one very important respect
- Lists are mutable
- Tuples and strings are immutable
- There are many operators that can be used to create objects of these immutable types, and variables can be bound to objects of these types
- However, objects of immutable types cannot be modified
- Objects of mutable types can be modified

Lists and tuples: mutable and immutable

- Lists differ from tuples and strings in one very important respect
- Lists are mutable
- Tuples and strings are immutable
- There are many operators that can be used to create objects of these immutable types, and variables can be bound to objects of these types
- However, objects of immutable types cannot be modified
- Objects of mutable types can be modified

Lists and tuples: mutable and immutable

- Lists differ from tuples and strings in one very important respect
- Lists are mutable
- Tuples and strings are immutable
- There are many operators that can be used to create objects of these immutable types, and variables can be bound to objects of these types
- However, objects of immutable types cannot be modified
- Objects of mutable types can be modified

Lists are mutable

The program below:

```
# Program to demonstrate the mutability of a list

L = [1, 2, 3, 4, 5]
print('L is:', L)

L[2] = 300
print('Now L is:', L)

print('Finished!')
```

produces the following output:

```
L is:  [1, 2, 3, 4, 5]
Now L is:  [1, 2, 300, 4, 5]
Finished!
```

Tuples are immutable

The program below:

```
# Program to demonstrate the immutability of a tuple

T = (1, 2, 3, 4, 5)
print('T is:', T)

T[2] = 300 # This won't work
print('Now T is:', T)

print('Finished!')
```

produces the following output:

```
Traceback (most recent call last):
  File "/home/john/Documents/dept/comp10280/2015/progs/p90.py",
            line 7, in <module>
    T[2] = 300
TypeError:  'tuple' object does not support item assignment
```

List Comprehension (1)

- List comprehension provides a concise way to apply an operation to the values in a sequence
- It creates a new list in which each element is the result of applying a given operation to a value from a sequence, eg the elements in another list
- For example, the program below:

```
# Program to demonstrate list comprehension

L = [x ** 2 for x in range(7)]
print('L is:', L)

print('Finished!')
```

produces the following output:

```
L is:  [0, 1, 4, 9, 16, 25, 36]
Finished!
```

List Comprehension (2)

- The `for` clause in a list comprehension can be followed by one or more `if` and `for` statements that are applied to the values produced by the `for` clause
- The additional clauses modify the sequence of values generated by the first `for` clause and produce a new sequence of values
- For example, the program below:

```
# Program to demonstrate a more complicated list comprehension

mixedList = [1, 2, 3.0, 'four', 5]
squaredList = [x ** 2 for x in mixedList if type(x) == int
               or type(x) == float]

print('mixedList is', mixedList)
print('squaredList is:', squaredList)
```

produces the following output:

```
mixedList is [1, 2, 3.0, 'four', 5]
squaredList is:  [1, 4, 9.0, 25]
Finished!
```

Program to count digits (1)

Initialuse the counter list
Prompt the user for a digit
Read digit
while *digit* $\geq$ *0 and digit* $\leq$ *9* ***do***
    if *digit == 0* ***then***
        increment the 0-counter
    else if *digit == 1* ***then***
        increment the 1-counter
    else if *digit == 2* ***then***
        increment the 2-counter
    . . .
    Prompt the user for another digit
    Read digit
Print out each of the counters
Program finishes

Program to count digits (2)

```
# Program to use a list to count the number of different digits entere

#Initialise the counter list
countList = [0 for x in range(4)]

# Prompt the user for a digit
number = int(input('Enter a digit between 0 and 3:  '))

while number >= 0 and number < 4:
    if number == 0:
        countList[0] += 1
    elif number == 1:
        countList[1] += 1
    elif number == 2:
        countList[2] += 1
    else:  # number == 3
        countList[3] += 1

# Prompt the user for another digit
    number = int(input('Enter a digit between 0 and 3:  '))

for i in range(4):
    print('Number of ', i, ':  ', countList[i])
```

Program to count digits (3)

```
# Program to use a list to count the number of different digits entere
# Uses the number as an index into the list

#Initialise the counter list
countList = [0 for x in range(10)]

# Prompt the user for a digit
number = int(input('Enter a digit between 0 and 9:  '))

while number >= 0 and number <= 9:
    countList[number] += 1

# Prompt the user for another digit
    number = int(input('Enter a digit between 0 and 9:  '))

for i in range(10):
    print('Number of ', i, ':  ', countList[i])

print('Finished!')
```