

Outline

[Arrays](#)

[Lists](#)

[Lists and strings: mutable and immutable](#)

[List Comprehension](#)

[Program to count digits](#)

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`
- 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

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

Lists

- A **list** in Python is an ordered sequence of elements
- The elements of a list can be of **any type** and need not be of the same type as each other
- Lists can be concatenated, indexed and sliced
- 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]`

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!
```

Python program to count numbers (1)

```
# Program to count numbers 0 - 3 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!
```

Python program to count numbers (2)

```
# Program to count numbers 0 - 3 using a list

# Initialise all list elements to 0
count = [0, 0, 0, 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)

```
# Program to count numbers 0 - 3 using a list

# Initialise all list elements to 0
count = [0, 0, 0, 0]

# Prompt the user for a number
number = int(input('Enter a number (an int >= 0 and <= 3): '))
while 0 <= number <= 3:
    count[number] += 1
# Prompt the user for another number
    number = int(input('Enter a number (an int >= 0): '))

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

Using the `for` statement on a list

The program below:

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

List = [1, 'two', 3.0]

print('List is:', List)
print('The elements of List are:')

for x in List:
    print(x)

print('Finished!')
```

produces the following output:

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

The program below:

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

List = [1, 'two', 3.0]

print('The elements of List are:')

for i in range(len(List)):
    print(List[i])

print('Finished!')
```

produces the following output:

```
List is:  [1, 'two', 3.0]
```

```
The elements of List are:
```

```
1
two
3.0
Finished!
```

Lists and strings: mutable and immutable

- Lists differ from strings in one very important respect
- Lists are **mutable**
- 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

You can change any element of a list

The program below:

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

```
L = [1, 2, 3, 4, 5]
```

```
print('L is:', L)
```

```
L[2] = 300 # changing element 3
```

```
print('Now L is:', L). # L[0] is element 1
```

```
print('Finished!')
```

produces the following output:

```
L is: [1, 2, 3, 4, 5]
```

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

```
Finished!
```

Strings are **immutable**

The program below:

```
# Program to demonstrate the mutability of a list  
s = 'ABBA'  
  
print('s is:', s)
```

Cannot write:

```
s[1] = 'a'
```

The following error arises:

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'str' object does not support item  
assignment
```

Cannot change the elements of a string

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)] # list comprehension
print('L is:', L)

print('Finished!')
```

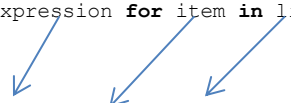
produces the following output:

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

List Comprehension (1)

General form of list comprehension:

```
[expression for item in list]
```



```
L = [x for x in range(10)] # list comprehension  
print('L is:', L)
```

produces the following output:

```
L is: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- 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)

Initialise the counter list

Prompt the user for a digit

Read digit

while *digit* ≥ 0 and *digit* ≤ 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 (3)

```
# Program to use a list to count the number of different digits entered
# 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!')
```

Outline

Operations on Strings and Lists


```
    ]  
countList = [0 for x in range(4)]
```

```
countList is now [0, 0, 0, 0]
```

While the above example is correct, there is a simpler solution to initialising a simple list:

```
countList = [0] * 4
```

```
countList is now [0, 0, 0, 0]
```

Operations on Strings and Lists

- We have now seen two different **sequence types**: `str`, and `list`
- `str`, and `list` have the following operations in **common**:
- `seq[i]` returns the *i*th element in the sequence
- `len(seq)` returns the length of the sequence
- `seq1 + seq2` returns the concatenation of the two sequences

Operations on Strings and Lists

- `n * seq2` returns a sequence that repeats `seq2` `n` times

e.g

```
str = "abcd"
```

```
2 * str is "abcdabcd"
```

- `seq[start:end]` returns a **slice** of the sequence from position `start` to `end` but NOT including `seq[end]`

e.g

```
str = "abcd"
```

```
str[0:2] is "ab"
```

```
L = [1, 2, 33, 4, 8, 6]
```

```
L[2:5] is [33, 4, 8]
```

- `e in seq` is `True` if `e` is contained in the sequence and `False` otherwise
- `e not in seq` is `True` if `e` is not in the sequence and `False` otherwise
- `for x in seq` iterates over the sequence

Methods

- In **Object-Oriented Programming (OOP)**, a **method** can be thought of as a function associated with a given **class**
- A **method invocation** can be thought of as the call/invocation of such a function to an **object** of that class
- We use **dot notation** to place the object to which the method is to be applied before the function name
- `o.m(args)` the method (function) `m` is called to operate on `o`

Methods associated with lists

- The following are some of the methods associated with lists
- All of them, except `count` and `index`, mutate the list
- `L.append(e)` adds the object `e` to the end of the list `L`
- `L.count(e)` returns the number of times that `e` occurs in `L`
- `L.insert(i, e)` inserts the object `e` into `L` at index `i`

Methods associated with lists

```
List = [1, 2, 3, 1, 16]
```

List.append(44) adds the object 44 to the end of List

Now List is [1, 2, 3, 16, 1, 44]

n = **List.count**(1) returns the number of times 1 occurs in List

n is now 2 because 1 occurs twice in List

Methods associated with lists

```
List = [1, 2, 3, 1, 16]
```

List.insert(2, 99) inserts the object 99 into List at index 2

Now List is [1, 2, **99**, 3, 16, 1, 44]

Methods associated with lists

L.extend(L1) adds the items in list L1 to the end of L

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

```
L1 = ['a', 'b', 'c' ]
```

```
L.extend(L1)
```

Now L is [1, 2, 3, 'a', 'b', 'c']

L.remove(e) deletes the first occurrence of e from L
(This method raises an **exception** if e is not in L)

```
L.remove(3)
```

Now L is [1, 2, 'a', 'b', 'c']

Methods associated with lists

L.index(e) returns the index of the first occurrence of e in L

(This method raises an **exception** if e is not in L)

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

```
i = L.index(3)
```

i is now 2 because 3 occurs at position 2

Methods associated with lists

`L.pop(i)` *removes and returns* the item at index `i` in `L`

If `i` is omitted, it defaults to `-1`, to remove and return the last element of `L`

```
L = [1, 2, 3, 4, 6]
```

```
x = L.pop()
```

Now `L` is `[1, 2, 3, 4]`

`x` is `6`

```
y = L.pop(0)
```

Now `L` is `[2, 3, 4]`

`y` is `1`

Methods associated with lists

L.reverse() reverses the order of elements in L

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

```
L.reverse()
```

Now L is [3, 2, 1]

Methods associated with lists

L.sort() sorts the elements in in L in ascending order

```
L = [1, 222, 3, 45, 6]
```

```
L.sort()
```

Now L is [1, 3, 6, 45, 222]

```
# Program to demonstrate methods on lists
a = [0, 1234, 2345, 77.96, 0, 2]
print('a is', a)

print('Number of occurrences of 77.96, 100 and 0: ',
      a.count(77.96), a.count(100), a.count(0))

a.insert(2, 100)
a.append(0)
print('a is', a)

print('First occurrence of 100 is at index', a.index(100))

a.remove(0)
print('a is', a)
a.reverse()
print('a reversed is', a)

a.sort()
print('a sorted is', a)

a.pop()
print('a, having popped the last element, is', a)
print('Finished!')
```

```
a is [0, 1234, 2345, 77.96, 0, 2]
Number of occurrences of 77.96, 100 and 0: 1 0 2
a is [0, 1234, 100, 2345, 77.96, 0, 2, 0]
First occurrence of 100 is at index 2
a is [1234, 100, 2345, 77.96, 0, 2, 0]
a reversed is [0, 2, 0, 77.96, 2345, 100, 1234]
a sorted is [0, 0, 2, 77.96, 100, 1234, 2345]
a, having popped the last element, is [0, 0, 2, 77.96, 100, 1234]
Finished!
```

Methods on strings (1)

- The following are some methods for strings
- Note that, since **strings are immutable**, all of them return values and have no side-effects
- `s.count(s1)` returns the number of times that the string `s1` occurs in `s`
- `s.find(s1)` returns the index of the first occurrence of the substring `s1` in `s`, and returns `-1` if `s1` does not occur in `s`
- `s.rfind(s1)` the same as `find`, but starts from the end of `s` (the “r” in `rfind` stands for “reverse”)

Methods on strings (3)

```
s = "ABBA"
```

```
t = s.lower() converts all uppercase letters in s to lowercase and stores them in t the string s is unchanged
```

```
t now contains abba
```

```
s = s.lower() converts all uppercase letters in s to lowercase
```

```
s now has value: abba
```

```
t = s.upper() converts all lowercase letters in s to uppercase and stores them in t
```

```
t now contains ABBA
```

Methods on strings (4)

```
s = "ABBA abba"
```

`s.replace(old, new)` returns list with all occurrences of the string `old` in `s` **replaced by** the string `new` stores them in `t`

Examples

```
t = s.replace('bb', 'xxxx')
```

`t` now contains ABBA axxxxa

```
s = s.replace('A', 'a')
```

`s` now contains aBBa abba

Methods on strings (5)

- `s.rstrip()` removes trailing whitespace from `s`
- **Whitespace** refers to the *space* character, *tab* character, *newline*, *return* character and *formfeed* i.e. characters that you cannot see on the screen
- Sometimes when you read a string the newline character will be part of the string at the end and you want to remove it.

Methods on strings (6)

- `s.split(d)` splits string `s` using `d` as a delimiter and returns the list of substrings making up `s`

If `d` is omitted, the substrings are separated by arbitrary strings of whitespace characters (space, tab, newline, return and formfeed)

This is a really useful and commonly used method

Methods on strings (6)

```
s = 'Joe, John, Bill, Mary'  
L = s.split(',')
```

We split the string using the comma character as delimiter

Now L is ['Joe', 'John', 'Bill', 'Mary']

```
s = 'Joe John Bill Mary'  
L = s.split()
```

We split the string using the space character as delimiter

Now L is ['Joe', 'John', 'Bill', 'Mary']

Methods on strings (6)

Say we have a string made up of a Name, rate of pay and hours worked. We can break the string into its components

```
s = 'Joe 10.5 40'  
L = s.split()
```

We can access the components in L

```
Name = L[0]  
Rate_per_hour = float( L[1])  
Hours_worked = float( L[2])
```

```
Name is now 'Joe'  
Rate_per_hour is now 10.5  
Hours_worked is now 40
```

Demonstrating methods on strings (1)

```
# Program to demonstrate methods on strings

a = 'Cristiano Ronaldo plays soccer with Portugal!'
print('a is:', a)
print('The length of a is:', len(a))

print('Number of occurrences of o:', a.count('o'))

print('First occurrence of o:', a.find('o'))
print('First occurrence of o, searching backwards:', a.rfind('o'))

print('String with all uppercase letters changed to lowercase:',
      a.lower())
print('a is:', a)

a = a.replace('Portugal', 'Manchester United')
print('a is:', a)

print('The words in a:', a.split(' '))

print('Finished!')
```

Demonstrating methods on strings (2)

```
a is: Cristiano Ronaldo plays soccer with Portugal!  
The length of a is: 45  
Number of occurrences of o: 5  
First occurrence of o: 8  
First occurrence of o, searching backwards: 37  
String with all uppercase letters changed to lowercase:  
    cristiano ronaldo plays soccer with portugal!  
a is: Cristiano Ronaldo plays soccer with Portugal!  
a is: Cristiano Ronaldo plays soccer with Manchester United!  
The words in a: ['Cristiano', 'Ronaldo', 'plays', 'soccer',  
                'with', 'Manchester', 'United!']  
Finished!
```


Demonstrating methods on strings (3)

```
# guess3.py: Guess the secret word
# Ignores case of words e.g. BLUE matches blue

secret = "Blue"
guess = " "
num_chances = 1
secret = secret.lower()      # convert to lowercase
while (guess != secret) and ( num_chances <= 3 ) :
    guess = input("Guess the secret word: ")
    guess = guess.lower()    # convert to lowercase
    if guess != secret:
        print("\nWrong guess: ", guess)
        num_chances = num_chances + 1
    else:
        print("Well done !")
if num_chances > 3:
    print("Sorry you have used all of your guesses")
    print("The secret word was: ", secret)
```

Running guess3.py:

Guess the secret word: **man**

Wrong guess: man

Guess the secret word: **dog**

Wrong guess: dog

Guess the secret word: **cat**

Wrong guess: cat

Sorry you have used all of your guesses

The secret word was: blue

Running guess3.py:

Guess the secret word: **black**

Wrong guess: black

Guess the secret word: **BLUE**

Well done !