
UNIVERSITÀ DI PADOVA


**Prof. Mauro Conti**

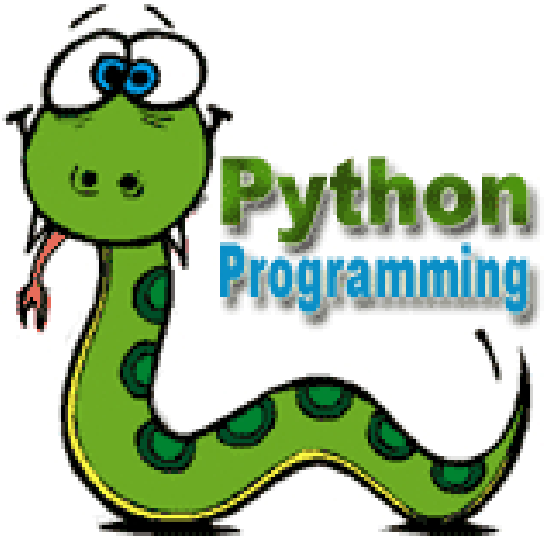
**T.A. : Hossein Fereidooni & Moreno Ambrosin**

< 2014 March >

# OUTLINE

**What you will become familiar with during the Python programming course are as follows:**

- Basic Operators
- Variable Types
- Numbers
- String
- Lists
- Tuples
- Dictionary
- Decision Making
- Loops
- Functions
- Modules
- Files I/O
- Exceptions
- Classes/Objects


Python
Programming

# A List is a kind of Collection

- A collection allows us to put many values in a single “variable”
- A collection is nice because we can carry all many values around in one convenient package.

*country = [‘USA', ‘Italy', ‘Iran' ]*

*greeting = [ ‘Hello', ‘Ciao', ‘Salam' ]*

# What is not a “Collection”

- Most of our variables have one value in them.
- when we put a new value in the variable, the old value is over written

```
>>> x = 2
>>> x = 4
>>> print x
4
```

# List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas.
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print [1, 24, 76]
[1, 24, 76]
>>> print ['red', 'yellow', 'blue']
['red', 'yellow', 'blue']
>>> print ['red', 24, 98.6]
['red', 24, 98.6]
>>> print [ 1, [5, 6], 7]
[1, [5, 6], 7]
>>> print []
[]
```

# Looking Inside Lists

- Just like strings, we can get at any single element in a list using an index specified in square brackets


```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print friends[1]
Glenn
```

# Lists are Mutable

- Strings are "immutable" - we *cannot* change the contents of a string - we must make a new string to make any change
- Lists are "mutable" - we *can* change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print x
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print lotto
>>>[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print lotto
[2, 14, 28, 41, 63]
```

# How Long is a List?

- The len() function takes a list as a parameter and returns the number of *elements* in the list
- Actually len() tells us the number of elements of *any* set or sequence (i.e. such as a string...)

```
>>> greet = 'Hello Bob'
>>> print len(greet)
9
>>> x = [ 1, 2, 'joe', 99]
>>> print len(x)
4
>>>
```

# Using the range function

- The range function returns a list of numbers that range from zero to one less than the parameter

```
>>> print range(4)
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print len(friends)
3
>>> print range(len(friends))
[0, 1, 2]
>>>
```

# Concatenating lists using +

- We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
>>> print a
[1, 2, 3]
```

# Lists can be sliced using :

- **Remember: *Just like in strings*, the second number is "up to but not including"**

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

# Building a list

- We can create an empty list and then add elements using the append method
- The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print stuff
['book', 99]
>>> stuff.append('cookie')
>>> print stuff
['book', 99, 'cookie']
```

# Is Something in a List?

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return True or False
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

# A List is an Ordered Sequence

- A list can hold many items and keeps those items in the order until we do something to change the order
- A list can be sorted. The sort method means "sort yourself"

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print friends
['Glenn', 'Joseph', 'Sally']
>>> print friends[1]
Joseph
>>>
```

# Built in Functions and Lists

- There are a number of functions built into Python that take lists as parameters

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print len(nums)
6
>>> print max(nums)
74
>>> print min(nums)
3
>>> print sum(nums)
154
>>> print sum(nums)/len(nums)
25
```

# Best Friends: Strings and Lists

```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print stuff
['With', 'three', 'words']
>>> print len(stuff)
3
>>> print stuff[0]
With
```

- *Split breaks a string into parts and produces a list of strings.*

# Best Friends: Strings and Lists

```
>>> line = 'A lot                of spaces'
>>> etc = line.split()
>>> print etc['A', 'lot', 'of', 'spaces']
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print thing
['first;second;third']
>>> print len(thing)
1
>>> thing = line.split(';')
>>> print thing
['first', 'second', 'third']
>>> print len(thing)
3
```

- *When you do not specify a delimiter, multiple spaces are treated like "one" delimiter.*
- *You can specify what delimiter character to use in the splitting.*

# The Double Split Pattern

- Sometimes we split a line and then grab one of the pieces of the line and split that piece again

```
>>> line="from hossein@math.unipd.it"
>>> mylist=line.split()
>>> print mylist
['from', 'hossein@math.unipd.it']
>>> myemail=mylist[1]
>>> print myemail
hossein@math.unipd.it
>>> myname=myemail.split('@')
>>> print myname
['hossein', 'math.unipd.it']
>>> print myname[0]
hossein
```

# Tuples

- Tuples are another kind of sequence that function much like a list
- They have elements which are indexed starting at 0

```
>>> x = ('Glenn', 'Sally', 'Joseph')
>>> print x[2]
Joseph
>>> y = ( 1, 9, 2 )
>>> print y
(1, 9, 2)
>>> print max(y)
9
```

# ..but.. Tuples are "immutable"

- Unlike a list, once you create a tuple, you cannot alter its contents - similar to a string

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print x
[9, 8, 6]
>>>
```

```
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback:'str'     object
does
not support item
Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback:'tuple' object
does
not support item
Assignment
>>>
```

# Things not to do with tuples

```
>>> x = (3, 2, 1)
>>> x.sort()
Traceback:AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback:AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback:AttributeError: 'tuple' object has no attribute 'reverse'
>>>
```

# Tuples and Assignment

- We can also put a tuple on the left hand side of an assignment statement

```
>>> (x, y) = (4, 'fred')
>>> print y
Fred
>>> (a, b) = (99, 98)
>>> print a
99
```

# Tuples are Comparable

- The comparison operators work with tuples and other sequences. If the first items are equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ( 'Jones', 'Sally' ) < ('Jones', 'Sam')
True
>>> ( 'Jones', 'Sally') > ('Adams', 'Sam')
True
```

# The dictionary data structure

- A dictionary is mapping between a set of indices (**keys**) and a set of **values**
  - The items in a dictionary are key-value pairs
- Keys can be any Python data type
  - Because keys are used for indexing, they should be immutable
- Values can be any Python data type
  - Values can be mutable or immutable

**{'one': 'uno' , 'two': 'due', 'three': 'tre'}**

# Creating a dictionary

```
>>>eng2it = dict()
>>>print eng2it
>>>eng2it['one'] = 'uno'
>>>print eng2it
>>>eng2it['two'] = 'due'
>>>print eng2it
```

# Creating a dictionary

```
eng2it = {'one': 'uno', 'two': 'due',  'three': 'tre'}
print eng2it
```

- In general, the order of items in a dictionary is unpredictable
- Dictionaries are indexed by keys, not integers

# Dictionary indexing

*print eng2it['three']*

*print eng2it['five']*

- If the index is not a key in the dictionary, Python raises an exception

# The `in` operator

- Note that the `in` operator works differently for dictionaries than for other sequences

  - For strings, lists, and tuples, *x in y* checks to see whether x is an *item* in the sequence

  - For dictionaries, *x in y* checks to see whether x is a *key* in the dictionary

# Tuples and Dictionaries

- The items() method in dictionaries returns a list of (key, value) tuples

```
>>> d = dict()
>>> d['two'] = 2
>>> d['four'] = 4
>>> tups = d.items()
>>> print tups
[('two', 2), ('four', 4)]
```

# Keys and values

- The `keys` method returns a list of the keys in a dictionary

*print eng2it.keys()*

- The `values` method returns a list of the values

*print eng2it.values()*

- The `items` method returns a list of tuple pairs of the key-value pairs in a dictionary

*print eng2it.items()*

# Sorting Lists of Tuples

- We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary
- First we sort the dictionary by the key using the items() method

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = d.items()
>>> t
[('a', 10), ('c', 22), ('b', 1)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

# REFERENCES


1. http://www.tutorialspoint.com/index.htm
2. http://docs.python.org/2/library/stdtypes.html
3. http://docs.python.org/lib/string-methods.html


# Contact

- **Website: http://www.math.unipd.it/~hossein/fereidooni.htm**
- **E-mail: hossein@math.unipd.it**
- **Skype: fereidooni1983**