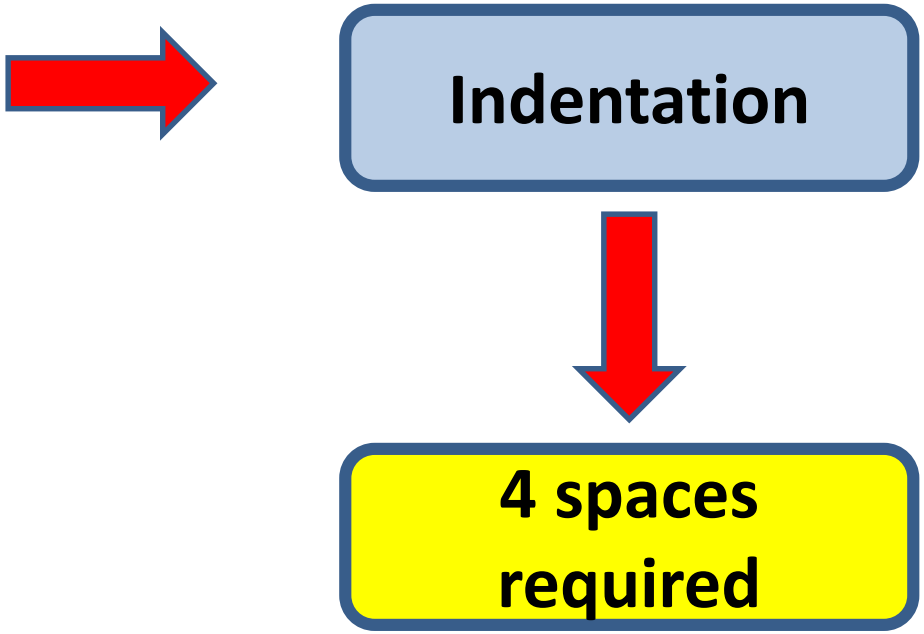
Intro to Python

Emmanouil Lakiotakis
17/03/2013

Basics

- Object-oriented programming language (e.g Java)
- No pointers (vs C)
- No arrays (instead lists)
- No brackets



Comments

- 1st method: # (single line comment)
- 2nd method: include your code between ‘’’ your code ‘’’ or “”” your code “”””
- Example:

```
def my_function(x, y):
    """This is the docstring. This
    function does blah blah blah."""
    # The code would go here...
```

Variable declaration

- The first assignment to a variable creates it.
  - Variable types don't need to be declared.
  - Python figures out the variable types on its own.
- Examples:
  - x = 34 - 23 (variable x is integer)
  - y = 23.3 (variable y is float)
  - my_string = "Hello world" (variable my_string is a string)

Variable type

- Command type(x) returns the data type of variable x
- Basic data types:
  - Integers (default for numbers)
    - z = 5 / 2 # Answer is 2, integer division.
  - Floats
    - x = 3.456
  - Strings (Can use “” or ‘’ to specify)
    - “abc” ‘abc’ (Same thing)
- Type cast: use int(), float() or str()

Basic operators

- Arithmetic Operators: (+, -,*, /, %, **)
- Comparison Operators: (==, !=, >, >=, <, <=)
- Assignment Operators: (=, +=, -=, …)
- Bitwise Operators: (&, |, …)
- Membership Operators: (in, not in)

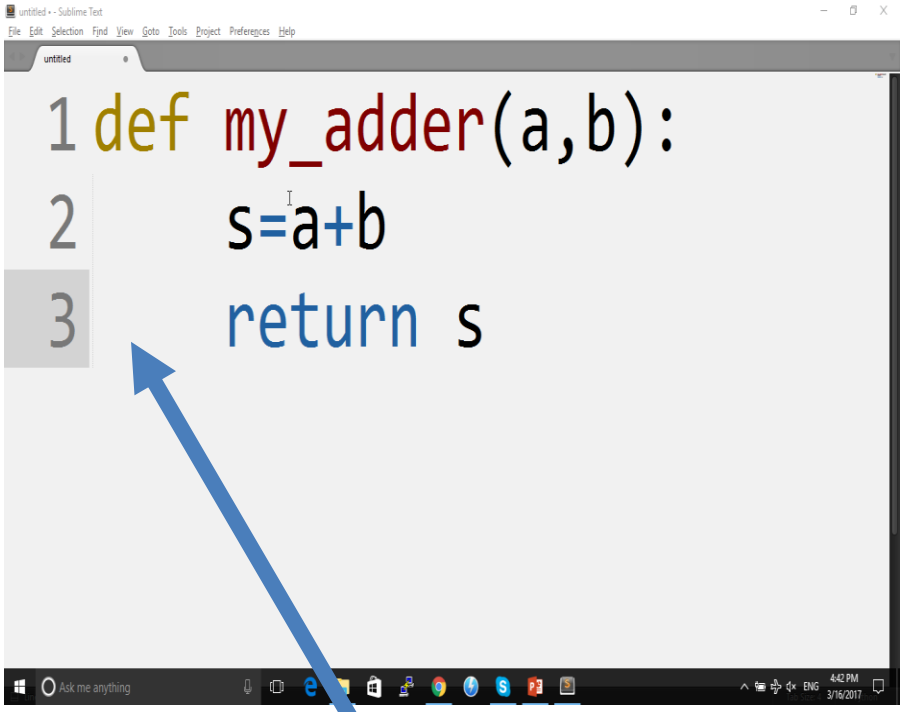
Function declaration

- def func_name(arg_1, arg_2, …, arg_n):

  (**4 spaces**)

  <python commands>

  (return result)



Function example

4 spaces

Strings

- String: a sequence of characters enclosed in **single** or **double** quotes
- Example: var1 = 'Hello World!‘
- Concatenation using + between strings
- String formatting (using % operand like C):
- >>print "My name is %s and weight is %d kg!" % ('Zara', 21)
- **Result**:
  My name is Zara and weight is 21 kg!

Python data structures

- **Sequence**: a collection of elements which each element of a sequence is assigned a number - its position or index starting from 0
- Well known sequence types:
  - Lists
  - Tuples
- Sequence operations:
  - Indexing
  - Slicing
  - Adding
  - Multiplying
  - Checking for membership

Lists (1/4)

- ***Mutable*** ordered sequence of items of **mixed types**
- List examples:
  - list1 = ['physics', 'chemistry', 1997, 2000]
  - list2 = [1, 2, 3, 4, 5, 6, 7]
  - list3 = ["a", "b", "c", "d"]
- **Attention**: list indices start at 0
- Accessing list elements:
  - print "list1[0]: ", list1[0] → list1[0]: physics
  - print "list2[1:5]: ", list2[1:5] → list2[1:5]: [2, 3, 4, 5]

Lists (2/4)

- Updating list elements
  - Example:
    list1 = ['physics', 'chemistry', 1997, 2000]
  - >> print list1[2] → 1997
  - >> list1[2] = 2001
  - print list1[2] → 2001
- Inserting values
  - list_name.insert(index,obj)
  - list_name.append()

Lists (3/4)

- Deleting elements
  - del list_name[indx]
  - list_name.remove(value)
- Length of a list: len(list_name)
- Concatenation of 2 lists: list1 + list2
- Membership: val **in** list_name (e.g 3 in [1, 2, 3])
- Iteration: for x **in** list_name
  Example: for x in [1, 2, 3]:

```
print x
```

Lists (4/4)

- Indexing - Slicing

  Example: L = ['spam', 'Spam', 'SPAM!']

  - L[2] → 'SPAM!'
  - L[-2] → 'Spam'
  - L[1:] → ['Spam', 'SPAM!']
- Useful methods:
  - max(list_name), min(list_name): return max or min element
  - list_name.reverse(): reverses list elements
  - list_name.sort([func]): sorts list elements
  - list(): convert in list

Tuples (1/4)

- Tuple: a sequence of **immutable** sequence of items
- Use parentheses, whereas lists use square brackets

Example:

```
tup1 = ('physics', 'chemistry', 1997, 2000)
```

- Attention: Any set of items, comma-separated, written without identifying symbols, i.e., brackets for lists, parentheses for tuples, etc., are regarded as **tuples**

Tuples (2/4)

- Accessing values:
- Example:

  tup1 = ('physics', 'chemistry', 1997, 2000)
- tup2 = (1, 2, 3, 4, 5, 6, 7)
  - >>print "tup1[0]: ", tup1[0] → physics
  - >> print "tup2[1:5]: ", tup2[1:5] → tup2[1:5]: [2, 3, 4, 5]

Tuples (3/4)

- Tuples are immutable: update or change the values of tuple elements is not allowed
- Concatenation is supported
- Removing individual tuple elements is not allowed only entire tuple with del command
- Also supported like lists:
  - Length: len()
  - Concatenation: +
  - Membership: in
  - Iteration: for x in tup

Tuples (4/4)

- Indexing and slicing is similar to lists
- Basic functions:
  - max(tup_name), min(tup_name): return max or min element from the tuple
  - tuple(): convert in tuple

Dictionaries (1/4)

- Dictionary: a structure with mapping between keys and values
- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces
- Each value has a unique key
- Keys must be immutable

Dictionaries (2/4)

- **Dictionary creation, insertion and update**

  Example:

  dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

  - >>print dict['Name'] ⇒ Zara
  - >>print dict['Age'] ⇒ 7
  - dict['Age'] = 8
  - >>print dict['Age'] ⇒ 8
  - dict['School'] = "DPS School“
  - >>print dict['School'] ⇒ DPS School

Dictionaries (3/4)

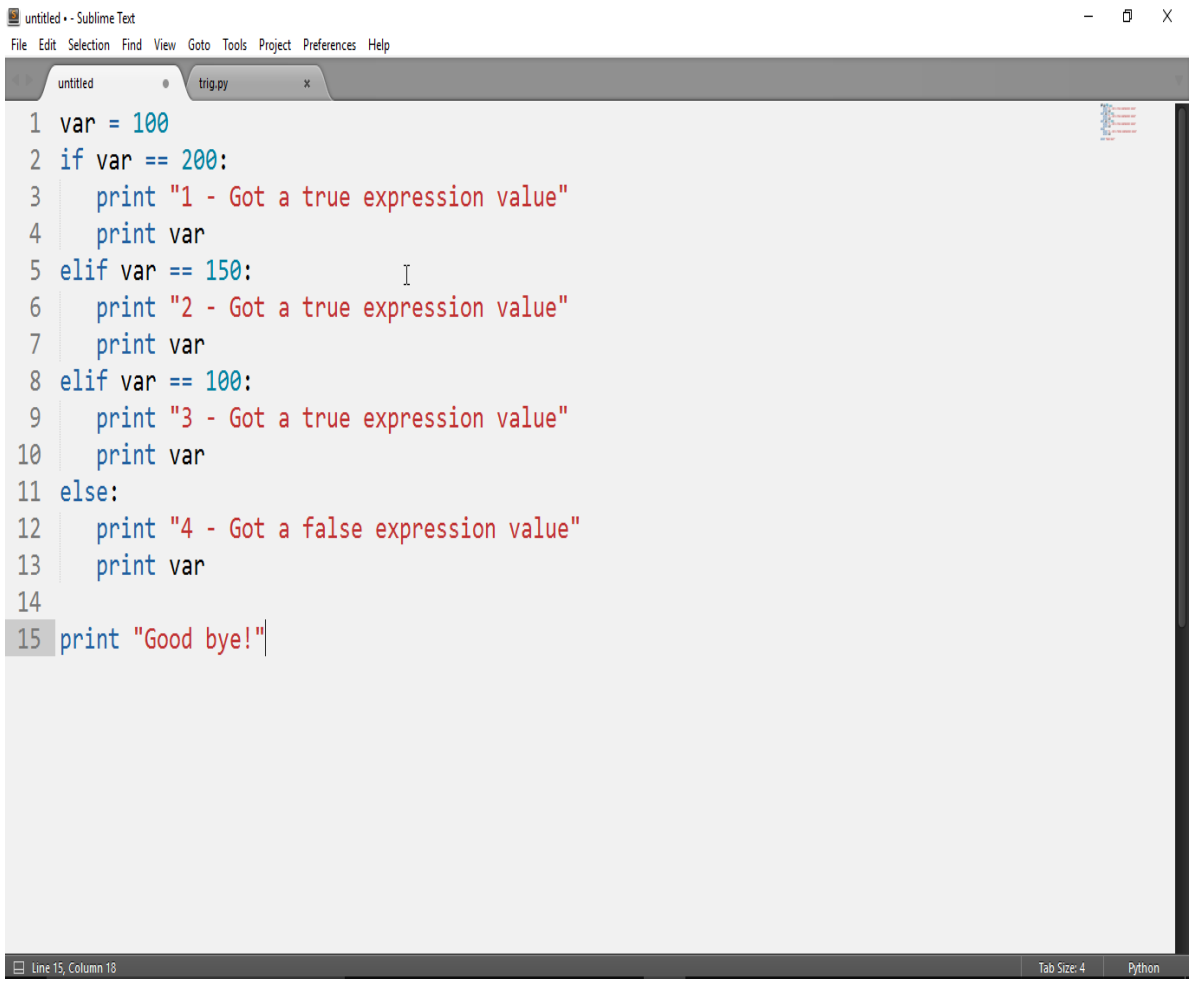
- Deletion
  - Individual dictionary elements:
    - del dict['Name']
  - Clear the entire dictionary
    - dict.clear()
  - Delete entire dictionary:
    - del dict

Dictionaries (4/4)

- Basic functions:
  - len(dict): returns the length of a dictionary
  - str(dict): string representation of a dictionary
  - dict.copy(): returns a copy of the dictionary
  - dict.has_key(key): returns True if dict has key as key
  - dict.items(): returns a list of dict's (key, value) tuple pairs
  - dict.keys(): returns list of dictionary dict's keys
  - dict.values(): Returns list of dictionary dict's values
  - dict.update(dict2): Adds dictionary dict2's key-values pairs to dict

Decision making

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```



Loops

- 3 basic types:
  - for
  - while
  - nested loops
- Loop Control Statements:
  - break: Terminates the loop statement
  - continue: skip the remainder of loop body
  - pass: when a statement is required syntactically but you do not want any command or code to execute.

For loop

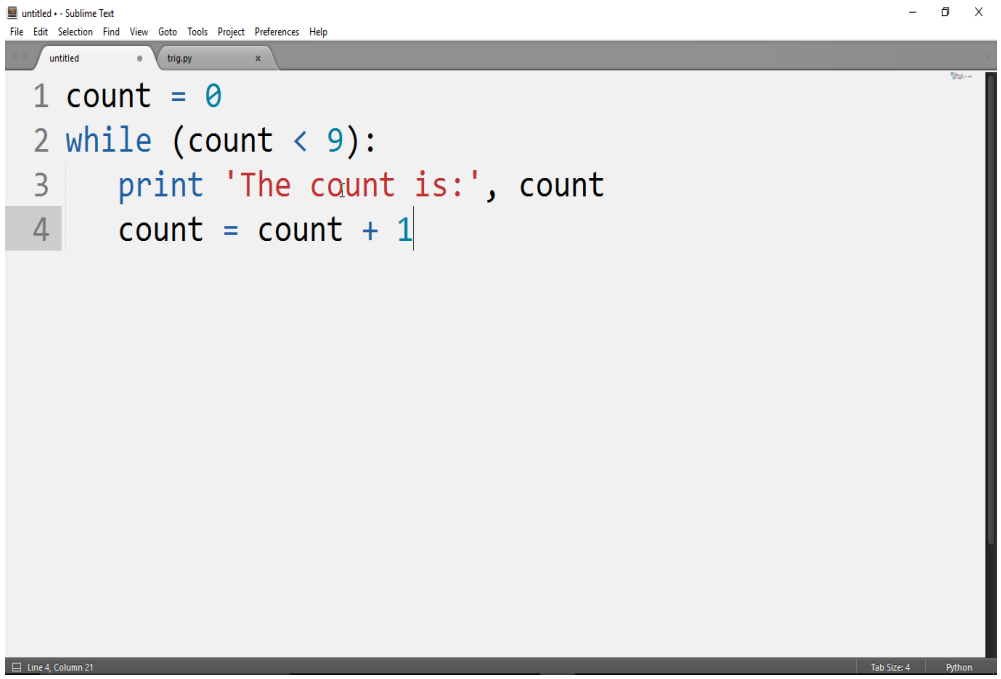
- 2 ways:
  - Iteration
  - Iteration by index
- range([start], stop[, step]):
  - start: Starting number of the sequence
  - stop: Generate numbers up to, **but not including this number**
  - step: Difference between each number in the sequence

```
for letter in 'Python':
    print 'Current Letter :', letter



fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
    print 'Current fruit :', fruits[index]
```

While loop

- while expression:

  statement(s)
- Don't forget indentation in for and while loops (4 spaces)
- There is no do … while syntax in python



Files I/O

- Open a file:
  - file object = open(file_name [, access_mode][, buffering])
- Close a file:
  - fileObject.close()
- Reading files
  - fileObject.read([num_of_bytes])
- Also useful:
  - fileObject.readlines()
- Writing files:
  - fileObject.write(string)

Other useful Topics

- Multi-threading programming in Python
- Call ping and traceroute commands via Python
  - **Help**: check subprocess.Popen()
- Socket programming

Question: How execute a Python script?

Answer: python my_script.py