Python Tutorial

Part I

Greg (mastergreg) Liras, John (nemo) Giannelos

foss.ntua

April 4, 2012

Outline

1. Introduction to Python
   - What is Python?
   - Freatures of Python
   - Why Python?
   - Dos and Don'ts
   - Variables and dynamic typing
2. Python Standard Types
   - Arithmetic
   - Strings
   - Data Structures
   - Functions
   - Epilogue

What is Python?

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Pythons elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Features (some of them)

In a few words, Python,

- is *Scripting Language*

Features (some of them)

In a few words, Python,

- is *Scripting Language*
- is *Strongly Typed*

Features (some of them)

In a few words, Python,

- is *Scripting Language*
- is *Strongly Typed*
- is *Dynamic*

Features (some of them)

In a few words, Python,

- is *Scripting Language*
- is *Strongly Typed*
- is *Dynamic*
- is *Interpreted*

Features (some of them)

In a few words, Python,

- is *Scripting Language*
- is *Strongly Typed*
- is *Dynamic*
- is *Interpreted*
- is *Portable*

Features (some of them)

In a few words, Python,

- is *Scripting Language*
- is *Strongly Typed*
- is *Dynamic*
- is *Interpreted*
- is *Portable*
- is *Object Oriented*

Features (some of them)

In a few words, Python,

- is *Scripting Language*
- is *Strongly Typed*
- is *Dynamic*
- is *Interpreted*
- is *Portable*
- is *Object Oriented*
- has *Vast Libraries (batteries included)*

Features (some of them)

In a few words, Python,

- is *Scripting Language*
- is *Strongly Typed*
- is *Dynamic*
- is *Interpreted*
- is *Portable*
- is *Object Oriented*
- has *Vast Libraries (batteries included)*
- is *Simple and non-obtrucive*

Why?

- It is easy to remember

Why?

- It is easy to remember
- You can develop rapidly

Why?

- It is easy to remember
- You can develop rapidly
- Readable Code (whitespace is semantically important!)

Why?

- It is easy to remember
- You can develop rapidly
- Readable Code (whitespace is semantically important!)
- Interface with C libraries

Bad Practices

- Inventing the wheel

Bad Practices

- Inventing the wheel
- One-liners - Obfuscated coding

Bad Practices

- Inventing the wheel
- One-liners - Obfuscated coding
- Having code on top level

Bad Practices

- Inventing the wheel
- One-liners - Obfuscated coding
- Having code on top level
- Huge imports

```
>>> from foo import *
```

Good Practices

- Search first code less

Good Practices

- Search first code less
- Use env to locate your interpreter

Good Practices

- Search first code less
- Use env to locate your interpreter
- Import only what you need

Good Practices

- Search first code less
- Use env to locate your interpreter
- Import only what you need
- Run pychecker on your code

Good Practices

- Search first code less
- Use env to locate your interpreter
- Import only what you need
- Run pychecker on your code
- ```
  if __name__ == "__main__":
      main()
  ```
  ```

Types

- x is just a name

```
>>> x = 1
>>> x = 'hello world'
```

Types

- x is just a name

```
>>> x = 1
>>> x = 'hello world'
```

- don't mix

```
>>> 'a'+1
TypeError: cannot concatenate 'str' and 'int' objects
>>> 'a'*3
'aaa'
```

Outline

1. Introduction to Python
   - What is Python?
   - Freatures of Python
   - Why Python?
   - Dos and Don'ts
   - Variables and dynamic typing
2. Python Standard Types
   - Arithmetic
   - Strings
   - Data Structures
   - Functions
   - Epilogue

Numeric types

- int (limitless :-D)

Numeric types

- int (limitless :-D)
- float (53 bits precision)

Numeric types

- int (limitless :-D)
- float (53 bits precision)
- complex $(1 + 2j)$

Operators

- $+$ (add)

Operators

- + (add)
- - (subtract)

Operators

- + (add)
- - (subtract)
- * (multiply)

Operators

- + (add)
- - (subtract)
- * (multiply)
- / (divide)

Operators

- + (add)
- - (subtract)
- * (multiply)
- / (divide)
- % (modulo)

Operators

- + (add)
- - (subtract)
- * (multiply)
- / (divide)
- % (modulo)
- = (assign)

Strings

- Strings are not lists! Strings are immutable!

Strings

- Strings are not lists! Strings are immutable!
- Simple concatenation:

```
>>> 'Hello' + 'World'
'HelloWorld'
```

Strings

- Strings are not lists! Strings are immutable!
- Simple concatenation:

```
>>> 'Hello' + 'World'
'HelloWorld'
```

- Slicing:

Strings

- Strings are not lists! Strings are immutable!
- Simple concatenation:

```
>>> 'Hello' + 'World'
'HelloWorld'
```

- Slicing:
  -
```
>>> 'HelloWorld'[0]
'H'
```

Strings

- Strings are not lists! Strings are immutable!
- Simple concatenation:

```
>>> 'Hello' + 'World'
'HelloWorld'
```

- Slicing:
  - `>>> 'HelloWorld'[0]`
    `'H'`
  - `>>> 'HelloWorld'[6:]`
    `'orld'`

Strings

- Strings are not lists! Strings are immutable!
- Simple concatenation:

```
>>> 'Hello' + 'World'
'HelloWorld'
```

- Slicing:
  - `>>> 'HelloWorld'[0]`
    `'H'`
  - `>>> 'HelloWorld'[6:]`
    `'orld'`
- Unicode Strings:

```
>>> ur'Hello\u0020World !'
u'Hello World !'
```

Lists

- ```
  >>> a = ['spam', 'eggs', 100, 1234]
  >>> a
  ['spam', 'eggs', 100, 1234]
  ```
  ```

Lists

- ```
  >>> a = ['spam', 'eggs', 100, 1234]
  >>> a
  ['spam', 'eggs', 100, 1234]
  ```
- Negative indices:

  ```
  >>> a[-2]
  100
  ```
  ```

Lists

- ```
  >>> a = ['spam', 'eggs', 100, 1234]
  >>> a
  ['spam', 'eggs', 100, 1234]
  ```
- Negative indices:
  ```
  >>> a[-2]
  100
  ```
- Concatenation:
  ```
  >>> a[:2] + ['bacon', 2*2]
  ['spam', 'eggs', 'bacon', 4]
  ```
  ```

Lists

- ```
  >>> a = ['spam', 'eggs', 100, 1234]
  >>> a
  ['spam', 'eggs', 100, 1234]
  ```
- Negative indices:
  ```
  >>> a[-2]
  100
  ```
- Concatenation:
  ```
  >>> a[:2] + ['bacon', 2*2]
  ['spam', 'eggs', 'bacon', 4]
  ```
- Comprehension:
  ```
  for i in a:
      print i
  ```
  ```

Tuples

- Immutable (just as strings)

Tuples

- Immutable (just as strings)
- Indexed

Tuples

- Immutable (just as strings)
- Indexed
- Nested

Sets

A set is an unordered collection with no duplicate elements.

Sets

A set is an unordered collection with no duplicate elements.

- ```
  >>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
  >>> set(basket)
  set(['orange', 'pear', 'apple', 'banana'])
  ```
  ```

Sets

A set is an unordered collection with no duplicate elements.

- ```
  >>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
  >>> set(basket)
  set(['orange', 'pear', 'apple', 'banana'])
  ```
- Operators:
  ```

Sets

A set is an unordered collection with no duplicate elements.

- ```
  >>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
  >>> set(basket)
  set(['orange', 'pear', 'apple', 'banana'])
  ```
- Operators:
  - a - b (in a but not in b)
  ```

Sets

A set is an unordered collection with no duplicate elements.

- ```
  >>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
  >>> set(basket)
  set(['orange', 'pear', 'apple', 'banana'])
  ```
- Operators:
  - a - b (in a but not in b)
  - a | b (in a or in b)
  ```

Sets

A set is an unordered collection with no duplicate elements.

- ```
  >>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
  >>> set(basket)
  set(['orange', 'pear', 'apple', 'banana'])
  ```
- Operators:
  - a - b (in a but not in b)
  - a | b (in a or in b)
  - a & b (in a and in b)
  ```

Sets

A set is an unordered collection with no duplicate elements.

- ```
  >>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
  >>> set(basket)
  set(['orange', 'pear', 'apple', 'banana'])
  ```
- Operators:
  - a - b (in a but not in b)
  - a | b (in a or in b)
  - a & b (in a and in b)
  - a ^b (in a or b but not in both)
  ```

Dictionaries

Maps of objects

Dictionaries

Maps of objects

- Easy to create

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Dictionaries

Maps of objects

- Easy to create

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

- Simple to use

```
>>> tel = dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
>>> tel['jack']
4098
```

To or not to return

- No return value ('None')

```
>>> def hi(s):
        print "hello",s
```

To or not to return

- No return value ('None')

```
>>> def hi(s):
        print "hello",s
```

- int or string?

```
>>> def add(a,b):
        if type(a)==int:
            return a+b
        else:
            return "not int"
>>> add(1,2)
3
>>> add('a',1)
'not int'
```

To or not to return

- No return value ('None')

```
>>> def hi(s):
        print "hello",s
```

- int or string?

```
>>> def add(a,b):
        if type(a)==int:
            return a+b
        else:
            return "not int"
>>> add(1,2)
3
>>> add('a',1)
'not int'
```

- lambdas

```
>>> add = lambda x,y : x+y
>>> add(1,2)
3
```

Questions??

Ask! :)

Thanks

- Thanks for watching
- Thanks to foss-ntua for hosting