

# Python 3

Divya Pai

Object Oriented Analysis and Design



# Python: History

- ◆ It was started in the late 80's, implementation started in 1989
- ◆ Version 2.0 on 16<sup>th</sup> October ,2000
- ◆ Development shifts to a more transparent, community backed process
- ◆ Version 3.0 on 3<sup>rd</sup> December, 2008
- ◆ Backwards incompatible with previous versions
- ◆ Some developers still choose to use Python 2.6

# What is Python?

- ◆ Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.
- ◆ Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines.
- ◆ Python is free to use, even for commercial products, because of its OSI-approved open source license.

# Definition

- ◆ Python also known as “Python 3000” and “Py3k” is a remarkably powerful dynamic programming language.
- ◆ It is the first intentionally backward compatible python release

# Why Python 3 ?

- ◆ Is less redundant
- ◆ Is less surprising
- ◆ Is more consistent
- ◆ Separates concerns more clearly
- ◆ It is production ready

# FEATURES

- ◆ A non-exhaustive list of features which are only available in 3.x releases and won't be back ported to the 2.x series:
- ◆ Strings are Unicode by default
- ◆ Clean Unicode/bytes separation
- ◆ Exception chaining
- ◆ Function annotations

# FEATURES

- ◆ Syntax for keyword-only arguments
- ◆ Extended tuple unpacking
- ◆ Non-local variable declarations
- ◆ Backward Compatibility

# Many New Features

- Python 3 introduces many new features

- Composite string formatting

- Dictionary comprehensions

```
a= {key.upper():value for key,value in d.items() }
```

- Function annotations

```
def square(x: int)→int:  
    return x*x
```

# DIFFERENCES

- ◆ Python 2:
  - ◆ “strings”
  - ◆ “unicode strings”
- ◆ Python 3:
  - ◆ “bytes”
  - ◆ “strings”

NOTE: Python 3 does not implicitly convert strings and bytes!!

# 2to3 tool

- ◆ It is a command line tool that helps convert python 2 code to python 3.
- ◆ It does not automate python2 to python 3.
- ◆ It's probably also fair to say that 3to2 is the road less traveled compared to 2to3 at this stage, so you might come across a few rough edges here and there. However, if you want to write 3.x code, it's definitely an idea worth exploring.

# Backwards Compatibility

- ◆ Backward compatible code is when we implement python3 in python2.
- ◆ An ‘unwritten law’ of software  
....and yet Python 3 breaks it- so badly that ‘Hello World’ no longer works! Lets see why...

## EXAMPLE CODE

```
$ python3 hello.py
```

```
File "hello.py", line 1
```

```
print 'Hello world'
```

```
Syntax error: Invalid syntax
```

# Backward Compatibility

- ◆ Here the codes of Python 2.x can be back ported to Python 3.x and vice versa by using 2to3 and 3to2 respectively
- ◆ But Python 3 has limitations as every input is taken in form of a string and so extra functions need to be called in order to evaluate it
- ◆ So one needs to be careful as Python 2 when back ported is bound to break the code!

# Redundancy: Iteration in Python

```
for character in string :  
    print character
```

```
for number in string :  
    print number
```

```
input file = open('foo.txt')
```

```
for line in input file:  
    print line
```

**One syntax** works for strings, lists and text files!

# Redundancy: Integer representation

- ◆ Java four primitive integer types, four corresponding wrapper class *BigInteger* class
- ◆ Python 2: two integer types - *int* & *long*
- ◆ Python 3: one integer type - *int*

# Redundancy: Object Model

- Two types of class in Python 2

```
Class Foo :          [Old style]
```

```
. . .
```

```
Class Bar (object) : [New style]
```

```
. . .
```

- Only 'new-style' classes exist in Python 3

# Redundancy: Console Object

- Python 2 has two functions providing console input:

*raw\_input* : yielding input as a string

*input* : yielding the result of calling *eval* on input

- Python 3 has one function **input**, with same behavior as that of **raw\_input**

# Syntax changes

- Python 2 print is a statement and python 3 print is a function

- Exception handling syntax is now:

*try:*

*except IOError as e:*

- Yes your code will break!

- Advantage:

- 1) Greater consistency with console input

- 2) Greater flexibility provided by keyword arguments

# Integer Division

- ◆ Python 2 (like C++ and java):

- ➔  $7 / 3 = 2$

- ◆ Python 3:

- $7 / 3 = 2.333$

- $7 // 3 = 2$

# Library Re organization

- ◆ The standard library has been cleaned up

- ◆ Especially network/internet modules

- ◆ Example Python 2:

```
from urllib2 import urlopen
u = urlopen("http://www.python.org")
```

- ◆ Example Python 3:

```
from urllib.request import urlopen
u = urlopen("http://www.python.org")
```

# Portability

- ◆ Assuming you can't find an alternative package that already supports Python 3, you still have a few options to consider:
- ◆ Port the library to 3.x. ("Porting" means that you make the library work on 3.x.)
- ◆ If that turns out to be really hard, and all your other dependencies do exist in 2.x, consider starting off in 2.x. As has already been explained in other places, good 2.x code will typically make switching painless as soon as every dependency has been successfully ported.

# Data Structure

- ◆ A "data structure" is a way of organizing data for efficient or easy access. This isn't directly relevant to data types. In many languages (like C++ or Java), defining a new class requires you to tell the compiler how an instance's members are laid out in memory, and things like that, but in Python you just construct objects and add members to them and the interpreter figures out how to organize them

# Data Structure cont.

- ◆ So, mapping dictionaries are data types, but they're also abstract data structures—and, under the covers, Python's dictionary objects are built from a specific concrete data structure (open-chained hash table with quadratic probing) which is still partly abstract (each bucket contains a duck-typed value)

# Data Structure contd...

- ◆ Things get blurry when you get to higher-level abstract data structures (like pointer-based nodes) and lower-level abstract data types (like order-preserving collection of elements that can do constant-time insertion and deletion at the head)

# The I/O Problem

- ◆ The io handling changes in python3 are the most problematic for porting
- ◆ Python 3 re-implements the entire IO stack
- ◆ IO handling issues cant be fixed by automatic code conversion tools (2to3)

# Memory Issue

- ◆ Text strings in Python 3 require either 2x or 4x as much as memory to store Python2
- ◆ Increased memory use does impact the performance of string operations that make copies of large substrings
  - ◆ Slices, joins, split, replace, strip etc.

Operations that process strings character often run at same speed

# Uses of Python 3

- ◆ Rapid prototyping
- ◆ Web scripting
- ◆ Ad hoc programming
- ◆ XML processing
- ◆ Database applications
- ◆ GUI applications
- ◆ Extention language

# Is there any documentation?

- References:

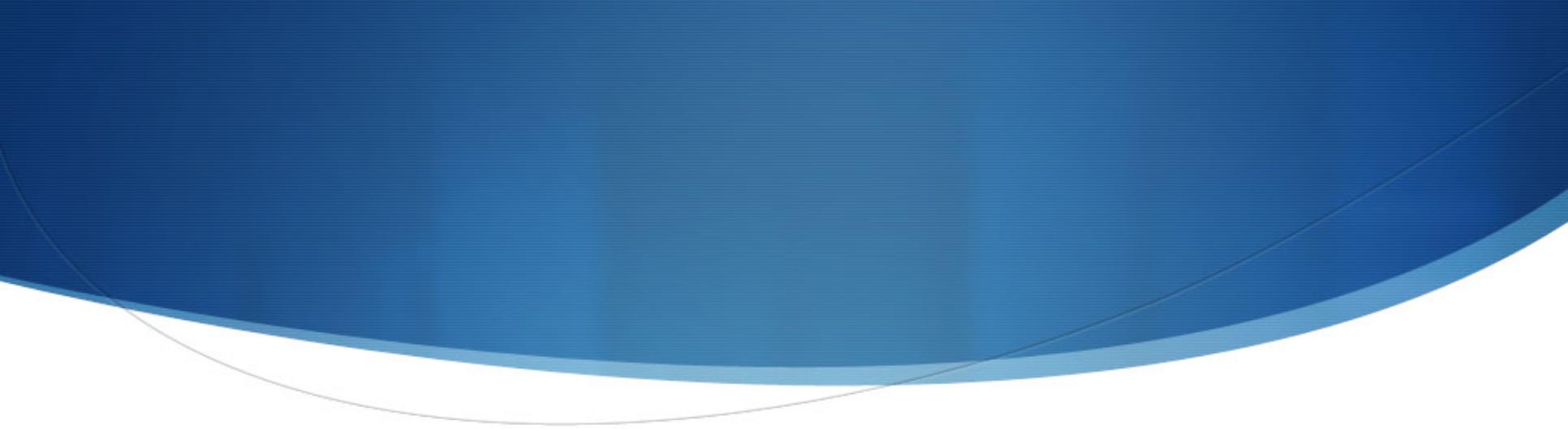
- Python website(<http://docs.python.org/3/>)

- Dive into Python(<http://diveintopython3.org>)

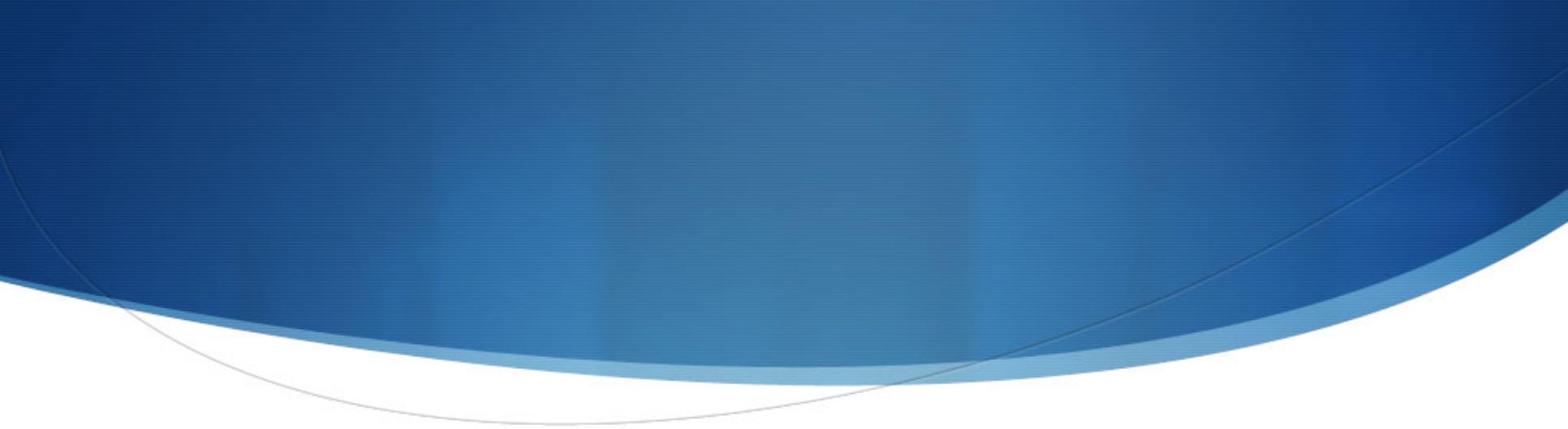
- Koders Code Search(<http://www.koders.com>)

- Unicode (<http://www.unicode.org/charts>)

- <http://www.slideshare.net/dabeaz/mastering-python-3-io>



QUESTIONS?

A decorative blue curved graphic at the top of the page, consisting of a dark blue outer arc and a lighter blue inner arc, both curving downwards.

THANK YOU

