IBM Software

AVP

Accelerated Value Program

ibm.com/software/support/acceleratedvalue/

AVP2929 – Jython DeepDive for WebSphere Application Server

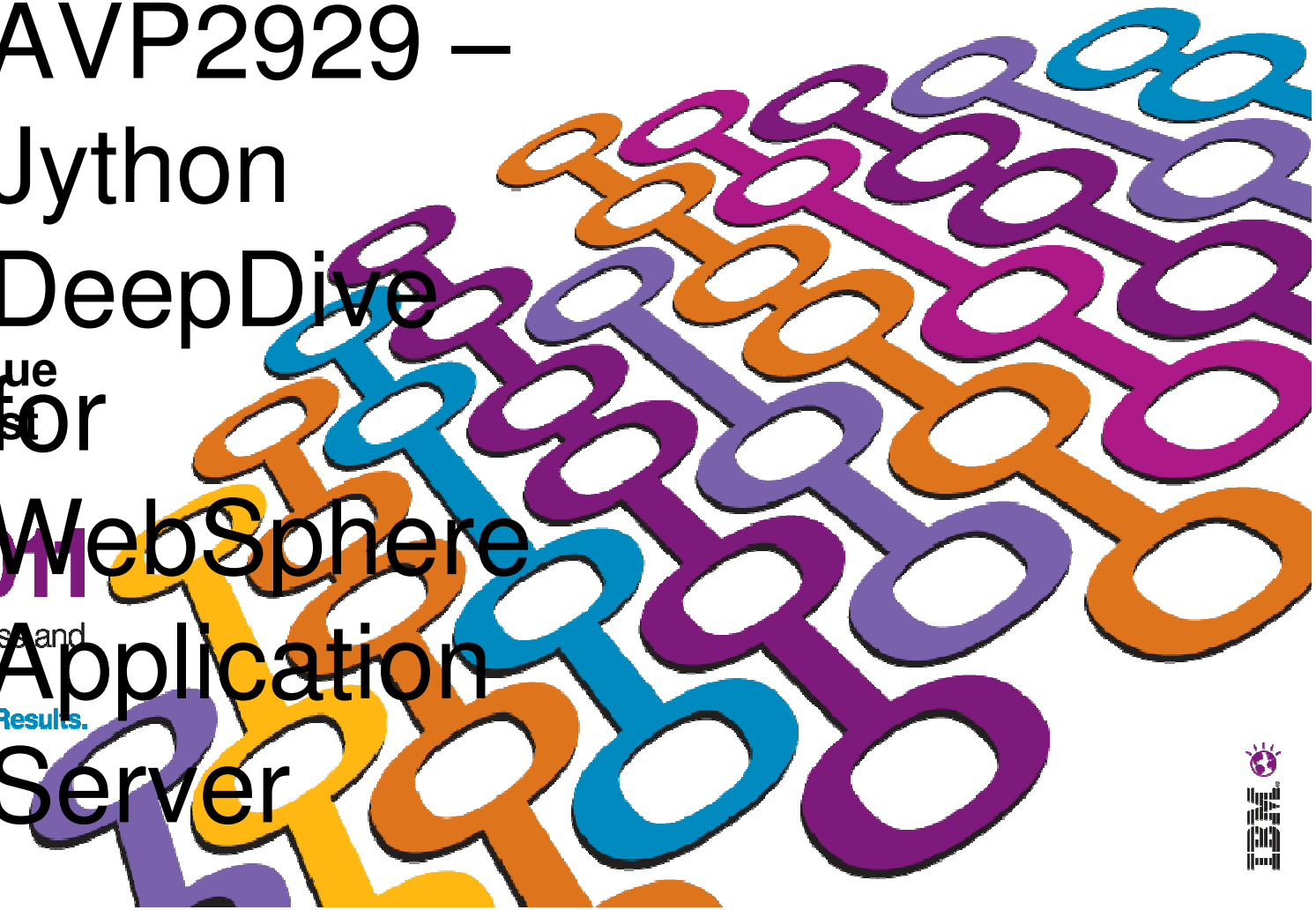
Benson Chen,
Accelerated Value Leader/Specialist

IBM Software

Impact2011

Changing the Way Business and IT Leaders Work

Optimize for Growth. Deliver Results.



Agenda

- Overview
- Jython DeepDive
- Using wsadmin for Configuration Scripting
- Introduction to the lab exercises

Overview

Jython Overview

- Jython = Python + Java
- Interpreted programming language (no compilation needed)
- Object oriented with ability to extend Java classes
- Interactive experimentation (wsadmin)
- Preferred programming language for WebSphere Application Server(WAS) automation

wsadmin Overview

- wsadmin is the scripting interface for WebSphere Application Server
- Gives the ability to manage, configure, deploy, and provide run-time operations
- Has the ability to do anything you can do in the console and more
- Scripting languages:
  - Jacl (default)
  - Jython (preferred)
- Executes a command or script or run in interactive mode

Jython DeepDive

IBM Software AVP

Jython Basics

- Comments start with # to end of line
- Variables untyped until assigned a value of some type using “=“ sign

  wsadmin>**name = "Billy"**

  wsadmin>**print name**

  Billy
- Statements exist in one logical line (can be used interactively in wsadmin)
- Can use ‘;’ to put multiple lines on one line of code

  wsadmin>**name = “Billy”; print name**

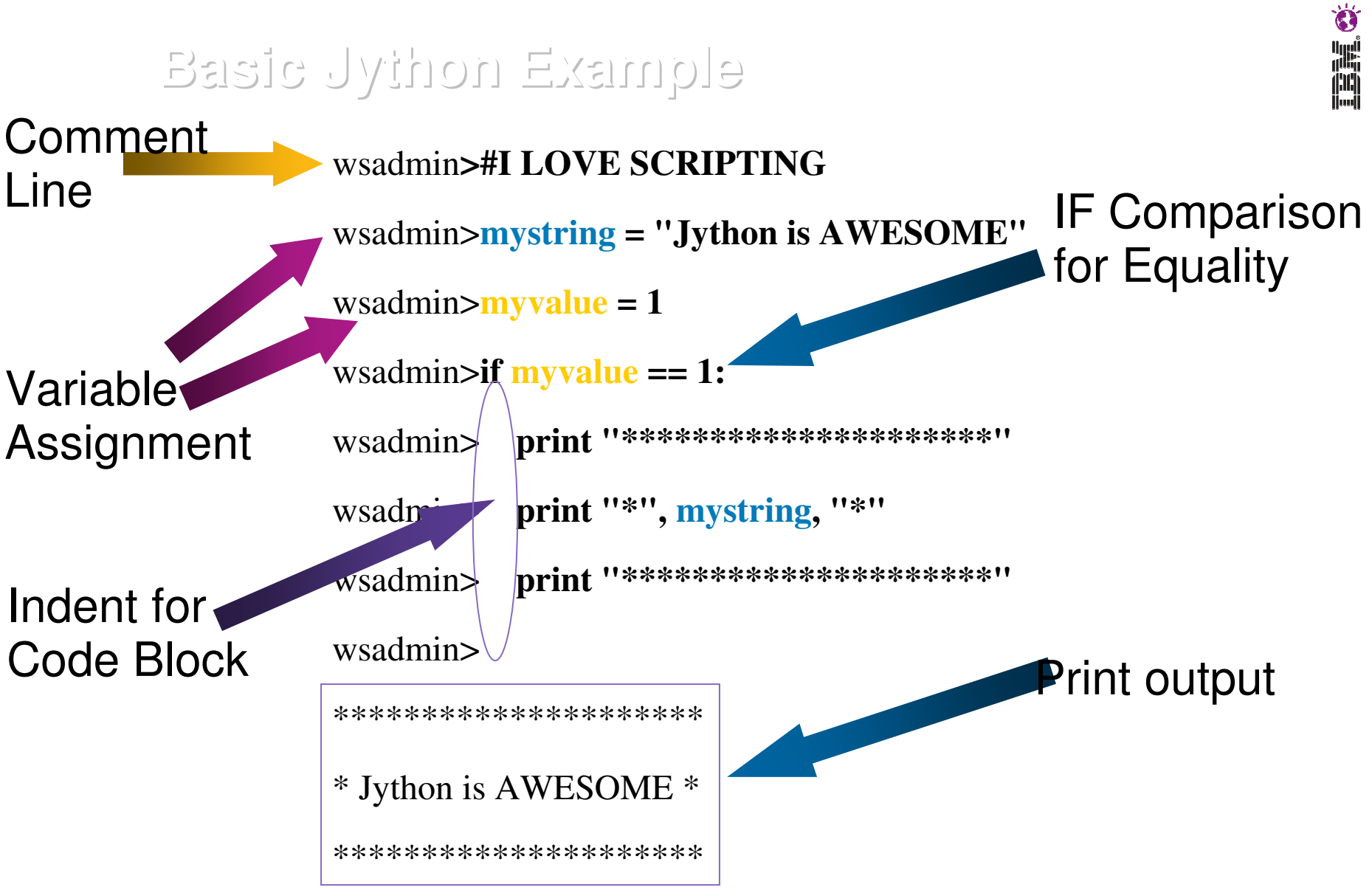
  Billy
- ‘\’ is used for line continuation of the same statement

  wsadmin>**print ”This is a very long sentence "**\

  wsadmin>**” that I will use to show continuation."**

  This is a very long sentence that I will use to show continuation.
- Code blocks are separated with indentation, can be tabs or spaces. Each statement within the code block MUST have the SAME indentation.

Basic Jython Example



Jython: Data Types

- String – Array of characters surrounded by single or double quotes

  wsadmin>**string = "Hello World"**

- Number – Commonly signed integers or floating point real values

  wsadmin>**digit = 1**

  wsadmin>**float = 10.0**

- List – Square brackets containing comma separated array of data (any type)

  wsadmin>**list = ["a", 2, "c", 4.1]**

  wsadmin>**listOfLists = [[], [1,2], [1,2,3]]**

  wsadmin>**listOfLists[0] = [5,6,7]**

  wsadmin>**print listOfLists**

  [[5,6,7], [1,2], [1,2,3]]

Jython: Data Types (cont.)

- Tuple – Same as List except use of parentheses and data is immutable (elements cannot change)

  wsadmin>**tuple = (1, “hello”, 2.3, (“a”, “b”), [1,2])**

  wsadmin>tuple[0] = 10

  WASX7015E: Exception running command: "tuple[0] = 10"; exception information:

  com.ibm.bsf.BSFException: exception from Jython:

  Traceback (innermost last):

  File "<input>", line 1, in ?

  TypeError: can't assign to immutable object

- Dictionary – Equivalent to Hashtable using curly braces and name:value pairs

  wsadmin>**employee = {'name':'Bob', 'id':'0001', 'dept':'sales'}**

  wsadmin>**print employee[’name’]**

  Bob

Jython String Formatting

- **Two approaches to form Strings with variables**
- **#1. Concatenate Strings and variables using ”+” sign. Convert non-Strings with str(<var>) function.**

  wsadmin>**name = ”Benson”**

  wsadmin>**account = 1234**

  wsadmin>**balance = 100.00**

  wsadmin>**print ”Account ”+str(account)+” owned by ”+name+” has $”+str(balance)**

  Account 1234 owned by Benson has $100.0

- **#2. Insert String substitutions (%s – String; %d – Digit; %f – Floating point)**

  wsadmin>**name = ”Benson”**

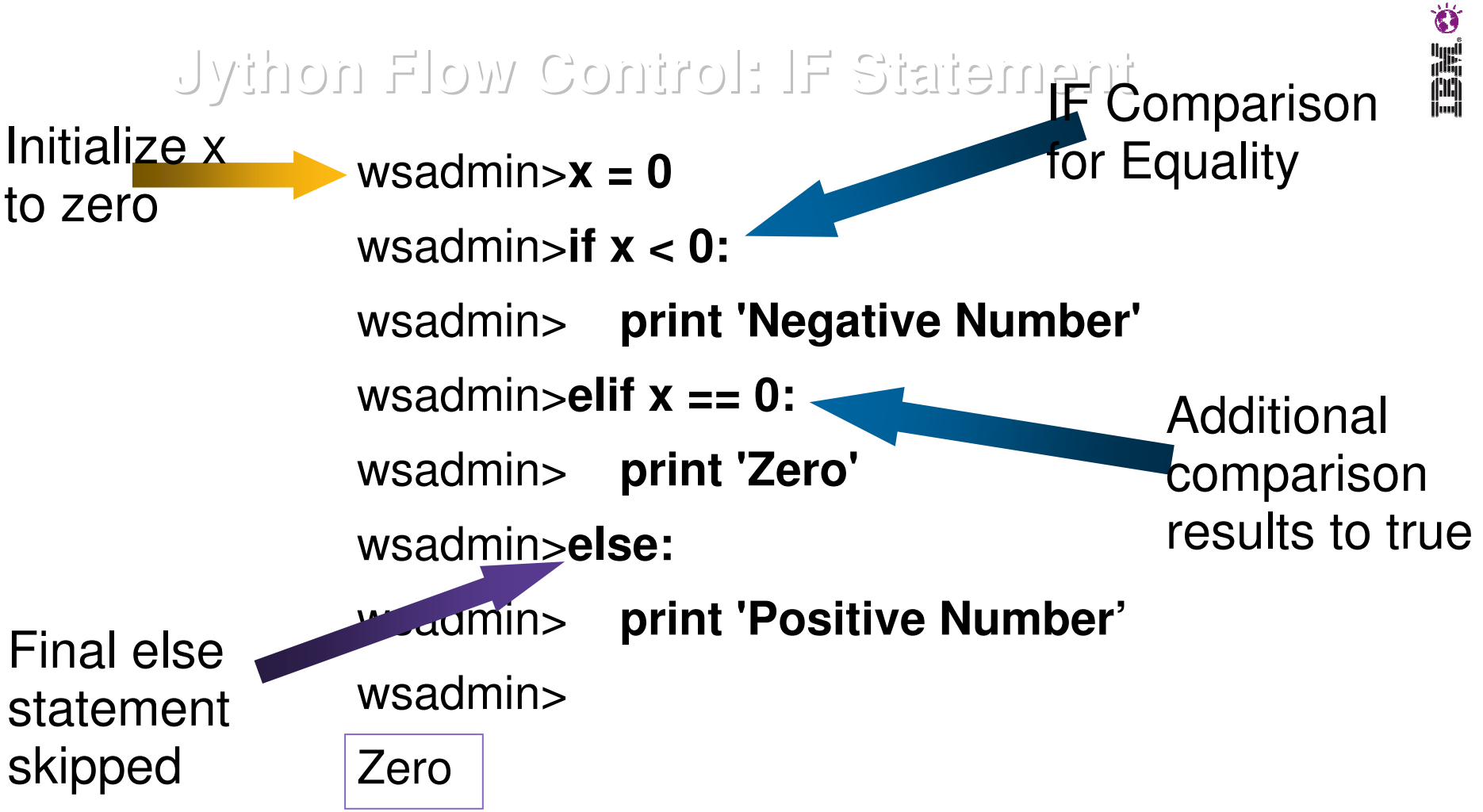
  wsadmin>**account = 1234**

  wsadmin>**balance = 100.00**

  wsadmin>**print ”Account %d owned by %s has $%f” % (account, name, balance)**

  Account 1234 owned by Benson has $100.000000

Jython Flow Control: IF Statement



Jython Flow Control: While Loop

Initialize x to zero

```
wsadmin>x = 0
wsadmin>while x < 30:
wsadmin>    x+=10
wsadmin>    print x
wsadmin>    if x > 100:
wsadmin>        break
```

```
10
20
30
```

While condition to continue

Increment and print for each loop

Safeguard IF condition to break loop

Jython Flow Control: For Loop

Initialize list with 3 values

```
wsadmin>list = [ 5, -1, 6 ]
wsadmin>for x in list:
wsadmin>    if x < 0:
wsadmin>        list.remove(x)
wsadmin>        print "Removed",x
wsadmin>
Removed -1
wsadmin>print list
[5, 6]
```

Iterate each item in list assign to x

Removes negative values

Jython Flow Control: Try/Except Statement

Initialize x and y

```
wsadmin>x = 1; y = 0
wsadmin>try:
wsadmin>    result = x / y
wsadmin>except ZeroDivisionError:
wsadmin>    print "division by zero!"
wsadmin>else:
wsadmin>    print "result is",result
wsadmin>
```

division by zero!

Try block to catch exceptions

Catch exception ZeroDivisionError

Run last if no exceptions caught

Jython: Common String Functions

- len(string)

  wsadmin>**x = "Hello World"**

  wsadmin>**len(x)**

  11

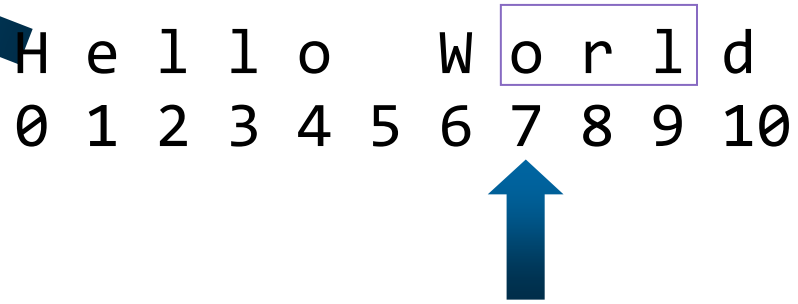
Last index
not included

- string.index(substring {, startIndex, endIndex})

  wsadmin>**x = "Hello World"**

  wsadmin>**x.index("orl", 5, len(x))**

  7



Jython: Common String Functions (cont.)

- string.split({separator}, {max_split})

```
wsadmin>x = "this is a test"
wsadmin>x.split(' ', 2)
['this', 'is', 'a test']
```

- Slicing string[{startIndex}: {endIndex}]

```
wsadmin>x = "Hello World"
wsadmin>x[1:5]
'ello'
wsadmin>x[:6]
'Hello '
wsadmin>x[6:]
'World'
```

Last index not included

```
H e l l o   W o r l d
0 1 2 3 4 5 6 7 8 9 10
```

Jython: Common List Functions

- len(list)

  wsadmin>**list = ["a", "b", "c"]**

  wsadmin>len(list)

  3

- Concatenate

  wsadmin>**[1, 2, 3] + [4, 5, 6]**

  [1, 2, 3, 4, 5, 6]

- Repetition

  wsadmin>**["a"] * 5**

  ['a', 'a', 'a', 'a', 'a']

- Membership

  wsadmin>**"a" in ["a", "b", "c"]**

  1 (true)

- Slicing list[{start},{end}]

  wsadmin>**list[0]**

  'a'

  wsadmin>**list[1:3]**

  ['b','c']

  wsadmin>**list[:2]**

  ['a','b']

- list.append(obj)

  wsadmin>**list = ["a", "b", "c"]**

  wsadmin>**list.append("d")**

  ["a", "b", "c", "d"]

- list.count(obj)

  wsadmin>**list = ["a", "b", "c"]**

  wsadmin>**list.count("b")**

  1

- list.index(obj)

  wsadmin>**list.index("c")**

  2

- list.insert(index, obj)

  wsadmin>**list.insert(2, "b-2")**

  ["a", "b", "b-2", "c"]

- list.pop()

  wsadmin>**list.pop()**

  'c'

- list.remove(obj)

  wsadmin>**list = ["a", "b", "c"]**

  wsadmin>**list.remove("a")**

  ["b", "c"]

- list.reverse()

  wsadmin>**list = ["a", "b", "c"]**

  wsadmin>**list.reverse()**

  ["c", "b", "a"]

Jython: Common Dictionary Functions

- len(dict)

  wsadmin>**employee = {'a':1, 'b':2, 'c':3}**

  wsadmin>**len(employee)**

  3

- dict.clear()

  wsadmin>**employee.clear()**

  {}

- dict.copy()

  wsadmin>**employee = {'a':1, 'b':2, 'c':3}**

  wsadmin>**acopy = employee.copy()**

  {'b':2, 'a':1, 'c':3}

- dict.has_key(key)

  wsadmin>**dict.has_key('a')**

  1 (true)

- dict.keys()

  wsadmin>**dict.keys()**

  ['b', 'a', 'c']

- dict.values()

  wsadmin>**dict.values()**

  [2, 1, 3]

Import os and sys modules

Import java.io as module name jio

```
import os; import sys
import java.io as jio

for arg in sys.argv:
    print arg

properties = java.util.Properties()
fis = jio.FileInputStream(sys.argv[0])
properties.load(fis)
```

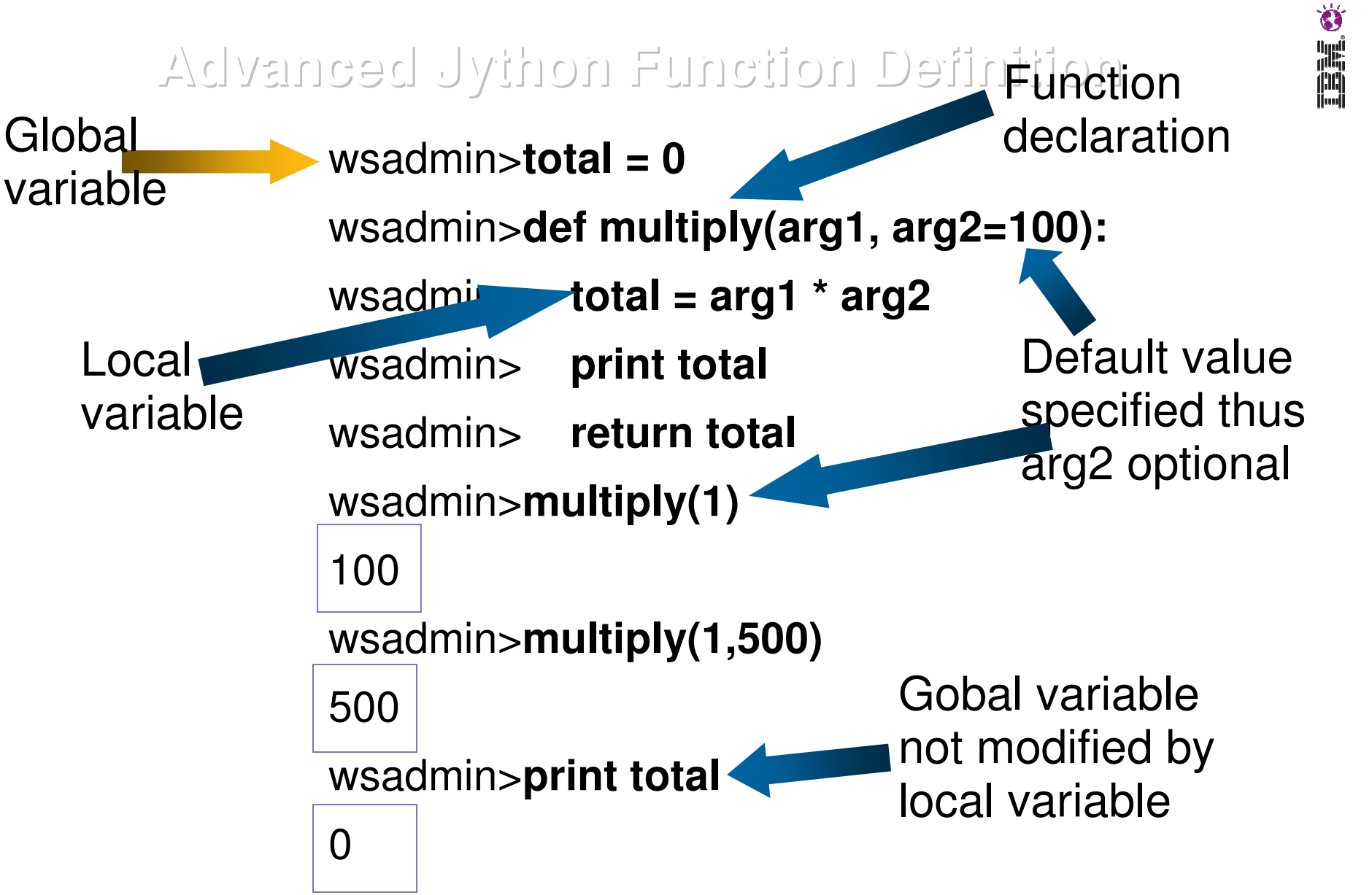
Command line arguments available as sys.argv list

Java constructors invoked without “new”

First argument providing file name

Populate properties from FileInputStream object

Advanced Jython Function Definition



Advanced Jython Using Global Variable

Global variable

```
wsadmin>total = 0
wsadmin>def multiplyG(arg1, arg2=100):
wsadmin>    global total
wsadmin>    total = arg1 * arg2
wsadmin>    print total
wsadmin>    return total
wsadmin>multiplyG(1)
```

Declare use of global variable

Global variable modified

```
100
wsadmin>multiplyG(1,500)
500
wsadmin>print total
500
```

Last function invocation modified global variable

Jython/Python References

- Jython Reference Book
  - http://www.jython.org/jythonbook/en/1.0/
- Python Reference Book
  - http://docs.python.org/library/index.html
- WAS V7 InfoCenter Jython Overview
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/cxml_jython.html
- DeveloperWorks Jython Tutorial
  - http://www.ibm.com/developerworks/java/tutorials/j-jython1/
  - http://www.ibm.com/developerworks/java/tutorials/j-jython2/

Using wsadmin for Configuration Scripting

wsadmin Usage

- **was_root/bin/wsadmin[.sh|.bat]**
  - **[-?]** – Provides help on full options
  - **[-c <commands>]** – Executes Jython or Jacl commands
  - **[-p <properties_file_name>]** – Uses specified properties
  - **[-f <script_file_name>]** – Executes .py or .jacl script file
  - **[-lang language]** – Specify jython or jacl (default) as script language
  - **[-conntype SOAP [-host host_name]**

    [-port port_number] [-user userid] [-password password] – Can specify remote DMGR via host and port. If security enabled, then user and password required.

wsadmin Tips

- **was_root/properties/wsadmin.properties**
  - Change the default script language to jython
    - **com.ibm.ws.scripting.defaultLang=jython**
  - Change the host and port if connecting to remote DMGR
    - **com.ibm.ws.scripting.port=8880**
    - **com.ibm.ws.scripting.host=localhost**
  - Enable tracing for debugging jython code
    - **com.ibm.ws.scripting.traceString=com.ibm.*=all=enabled**
- The DMGR must be running in order to execute configuration changes
- Readline Wrapper (rlwrap) for command line history
  - Google rlwrap and compile for your distribution
  - NOTE: Windows wsadmin.bat already has command line history
  - **rlwrap –r ./wsadmin.sh**

wsadmin Jython Objects (WAS 6.1 and 7.0)

- **AdminControl** - Used to run operational commands
  - Examples: Start/Stop server or enable/disable tracing
- **AdminConfig** - Used to run configuration commands to create or modify WebSphere Application Server configuration elements
  - Examples: Modify JVM properties or create JDBC datasource
- **AdminApp –** Used to install, modify, and administer applications
  - Examples: Deploy application and modify app properties
- **AdminTask -** Used to run administrative commands
  - Examples: Helper tasks for various admin tasks
- **Help -** Used to obtain general help

Help

- **print Help.help()** – Prints out functions available in the Help object.
- **print Help.message("ADMU3000I")** – Describes an error message
- **print AdminConfig.help()** – Prints out the functions available from the AdminConfig object
- **print AdminConfig.help(“getid”)** – Specify a function to get usage parameters. For example, “getid” function.

WASX7085I: Method: getid

Arguments: containment path

Description: Returns the configuration ID for an object described by the

given containment path -- for example, /Node:myNode/Server:s1/JDBCProvider:jdbc1/

Containment Path getid conversion to Configuration ID

- **Containment Path** is a hierarchical XPATH to easily describe a resource
  - **Format:** /type:name/type:name/type:name/.../
  - **Example:** /Cell:myCell/Node:myNode/Server:server1
  - **NOTE**: Only need to specify portion that is unique. If server name is uniquely named then use:

    /Server:myUniqueServerName
- **AdminConfig.types()** – Shows all the different types
- Resulting **AdminConfig.getid(containment_path)** provides the Configuration ID used for other AdminConfig functions:
  - server1(cells/myCell/nodes/myNode/servers/server1|server.xml#Server_1287418657728)

List resources of a given type

- **AdminConfig.list(<type>)** – Returns ALL Configuration IDs of this type of resource in one string separated by newline
  - **print AdminConfig.list("ProcessDef")**

    '(cells/cellname/nodes/nodename/servers/dmgr|server.xml#Java ProcessDef_1)

    ….

    <configidN>'
- Use split("\n") to split output into array List
  - **procList = AdminConfig.list("ProcessDef").split("\n")**
    - ['<configid1>', …., '<configidN>']
- Iterate resulting list with for loop
  - **for proc in procList:**

    print proc

List resources of a given type filtered by scope

- **AdminConfig.list(<type>, <scope_configid>)** – Returns ALL Configuration IDs of this type of resource scoped to an owning resource
  - **server1 = AdminConfig.getid("/Server:server1")**
  - **print AdminConfig.list("ProcessDef", server1)**

    '(cells/cellname/nodes/nodename/servers/server1|server.xml#JavaProcessDef_1)'
- **AdminConfig.list(<type>, <pattern_configid>)** – Returns ALL Configuration IDs of this type of resource scoped to an owning resource name's pattern
  - **print AdminConfig.list("ProcessDef", "*/server1|*")**

    '(cells/cellname/nodes/nodename/servers/server1|server.xml#JavaProcessDef_1)'

Show attributes of a configuration resource

- **AdminConfig.show(<configid>)** – Shows ALL attributes of the configuration resource
  - **server1 = AdminConfig.getid("/Server:server1")**
  - **server1Attrs = AdminConfig.show(server1); print server1Attrs**

    [customServices []]

    [developmentMode false]

    [errorStreamRedirect
    (cells/benuntuCell01/nodes/benuntuNode01/servers/server1|server.xml#StreamRedirect_1287418657730)]

    [name server1]

    [processDefinitions
    [(cells/benuntuCell01/nodes/benuntuNode01/servers/server1|server.xml#JavaProcessDef_1287418657731)]]

    ....

Show single attribute

- **AdminConfig.showAttribute(<configid>, <attrName>)** – Shows a single attribute of the configuration resource
  - **server1 = AdminConfig.getid("/Server:server1")**
  - **AdminConfig.showAttribute(server1, "processDefinitions")**

    [(cells/benuntuCell01/nodes/benuntuNode01/servers/server1|server.xml #JavaProcessDef_1287418657731)]
- The output appears to be a List but it is actually a String. Need to strip the square brackets
  - **AdminConfig.showAttribute(server1, "processDefinitions")[1:-1]**

    (cells/benuntuCell01/nodes/benuntuNode01/servers/server1|server.xml# JavaProcessDef_1287418657731)
- If multiple items separated by space, use split(" ") function to create a real List that you can iterate
  - **procDefList = AdminConfig.showAttribute(server1, "processDefinitions")[1:-1].split(" ")**

Create a resource

- **AdminConfig.required(<type>) –** Shows the required attributes to create a resource of a given type
  - **print AdminConfig.required("JDBCProvider")**

    Attribute Type

    name String

    implementationClassName String
- **AdminConfig.create(<type>, <parent_configid>, <attrs>) –** Creates a resource of a given type under a specific parent using a list of attributes (format: [[attr1 val1] [attr2 val2]… [attrN valN]])
  - **cell = AdminConfig.list("Cell").split("\n")[0]**
  - **AdminConfig.create("JDBCProvider", cell, "[[name testJDBCProvider] [implementationClassName com.test.jdbc.myImpl]]")**

Modify a resource attributes

- **AdminConfig.modify(<configid>, <attrs>) –** Modifies the attributes of a resource
  - **jdbcProv = AdminConfig.getid("/JDBCProvider:testJDBCProvider")**
  - **print AdminConfig.show(jdbcProv)**

    [classpath []]

    [implementationClassName com.test.jdbc.myImpl]

    [name testJDBCProvider]
  - **AdminConfig.modify(jdbcProv, "[[implementationClassName com.test.jdbc.otherJDBC] [description 'Best JDBC Provider']]")**
  - **print AdminConfig.show(jdbcProv)**

    [classpath []]

    [description "Better JDBC Provider"]

    [implementationClassName com.test.jdbc.otherJDBC]

    [name testJDBCProvider]
- **AdminConfig.unsetAttributes(<configid>, <attrNames>) –** Reset to default value or if no default, remove the value.
  - **AdminConfig.unsetAttributes(jdbcProv, "[[description] [classpath]]")**

Remove a resource

- **AdminConfig.remove(<configid>) –** Removes a resource specified by the configid
  - **jdbcProv = AdminConfig.getid("/JDBCProvider:testJDBCProvider")**
  - **AdminConfig.remove(jdbcProv)**

AdminTask helper commands

- **AdminTask.help() –** Shows the help for the AdminTask object
- **AdminTask.help("-commands") –** Lists all the helper commands available for the AdminTask object
- **AdminTask.help("-commands", <pattern>) –** Lists the helper commands that match the specified pattern
  - **print AdminTask.help("-commands", "*JDBC*")**

    WASX8004I: Available admin commands:

    createJDBCProvider - Create a new JDBC provider that is used to connect with a relational database for data access.

    listJDBCProviders - List the JDBC providers that are contained in the specified scope.
- **AdminTask.help(<commandName>) –** Shows the help information for using the specified command
  - **print AdminTask.help("createJDBCProvider")**

AdminTask invoke command

- **AdminTask.<command>(‘[-parm1 “value1” –parm2 “value2” … -parmN “valueN”]’) –** Invokes a command with set of parameter/value pairs within a square bracketed String. Invoking help on the command will list the required parameters with an asterisk in front of the parameter name.
  - **AdminTask.createJDBCProvider(‘[-scope Cell=myCell –databaseType DB2 –providerType “DB2 Universal JDBC Driver Provider” –implementationType “XA data source” –name “myDb2JDBCProvider”]’)**
- **AdminTask.<command>('-interactive') –** Invokes the command in interactive mode where you are prompted to enter the value for each parameter.
  - **AdminTask.createJDBCProvider('-interactive')**

Script Libraries (V7+ Only)

New to WAS V7 is a compilation of script libraries that can expedite certain tasks with one single command. Overlaps and contains more functions than the AdminTask object. Script files automatically loaded from <was_root>/scriptLibraries/.

Category	Jython Script File
Application	AdminApplication AdminBLA
Resources	AdminJ2C AdminJDBC AdminJMS AdminResources
Security	AdminAuthorizations
Servers	AdminClusterManagement AdminServerManagement
System	AdminNodeGroupManagement AdminNodeManagement
Application	AdminApplication AdminBLA
Utilities	AdminLibHelp AdminUtilities

Invoke Script Library function

- **<ScriptLibraryFile>.help() –** Each Script Library file contains a help() function to describe available functions.
  - **print AdminJDBC.help()**
- **<ScriptLibraryFile>.help(<functionName>) –** Specify a function name to print usage of the function
  - **print AdminJDBC.help("createJDBCProvider")**
- **<ScriptLibraryFile>.<function>(<arg1>, <arg2>,…,<argN>) -** Each function requires different arguments, please examine the help for the particular function for more details.
  - **AdminJDBC.createJDBCProvider("myNode", "myServer", "myJDBCProvider", "com.test.jdbc.MyImpl")**

Save or reset configuration changes

- **AdminConfig.save() –** Invokes the save command to preserve all the changes that has been made thus far
- **AdminConfig.setSaveMode(<mode>) –** Sets the directive if there is a configuration conflict. The two possible modes are:
  - **"rollbackOnConflict" –** (Default) Cause save operation to fail if changes conflict with other configuration changes
  - **"overwriteOnConflict" -** Save changes even if they conflict with other configuration changes
- **AdminConfig.reset() –** Invokes the reset command to restore the configuration to the last saved configuration. All changes made since the last save will be lost.

wsadmin References

- WAS V7 InfoCenter wsadmin Topic
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6topscripting.html
- WAS V7 Admin Scripting Redbook
  - http://publib-b.boulder.ibm.com/abstracts/redp4576.html?Open
- Whitepaper: Using Jython Scripting Language With WSADMIN
  - http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100963
- Developerworks Sample Scripts for WSADMIN
  - http://www.ibm.com/developerworks/websphere/library/samples/SampleScripts.html

Introduction to lab exercises

Lab Overview

- Basic tasks using wsadmin
- Exploring the scripting libraries
- Console Command Assistance
- Writing Jython scripts
- Tools for writing and executing scripts
- Debugging scripts

We value your feedback

- Please complete the survey for this session
- Hands-on Lab Jython Tools for Scripting

Thank You…