Python Training

A basic overview

IGATE

Speed.Agility.Imagination

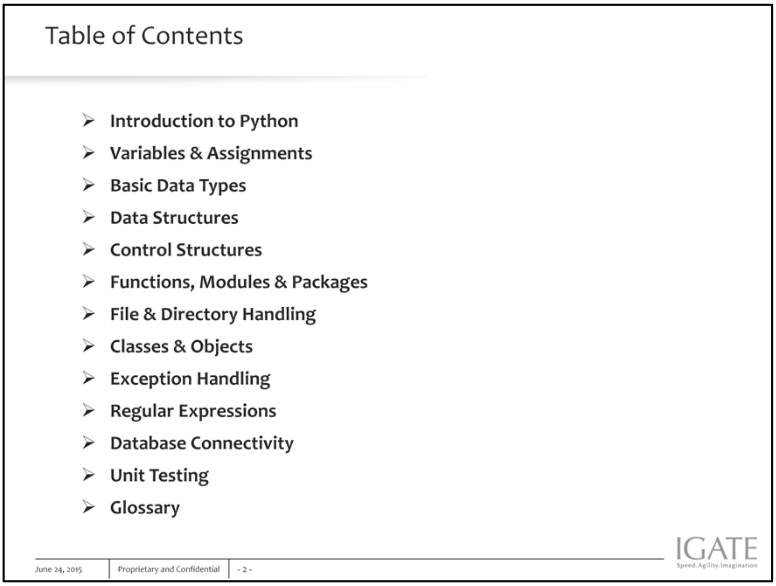
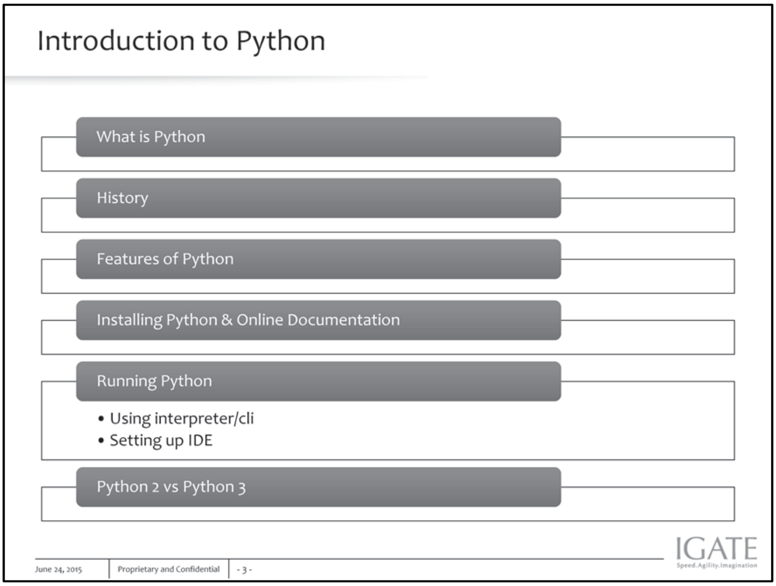


Table of Contents

- Introduction to Python
- Variables & Assignments
- Basic Data Types
- Data Structures
- Control Structures
- Functions, Modules & Packages
- File & Directory Handling
- Classes & Objects
- Exception Handling
- Regular Expressions
- Database Connectivity
- Unit Testing
- Glossary

June 24, 2015 | Proprietary and Confidential | - 2 -

IGATE
Speed.Agility.Imagination

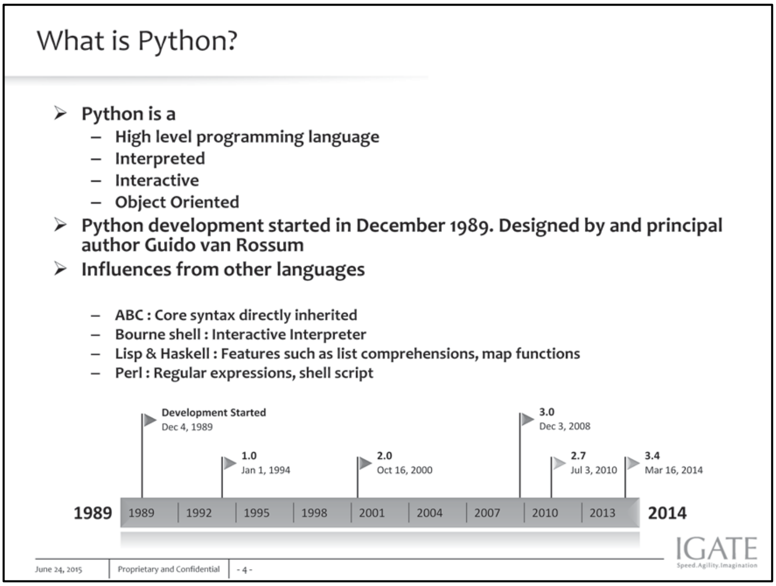


Introduction to Python

- What is Python
- History
- Features of Python
- Installing Python & Online Documentation
- Running Python
  - Using interpreter/cli
  - Setting up IDE
- Python 2 vs Python 3

June 24, 2015 | Proprietary and Confidential | - 3 -

IGATE
Speed.Agility.Imagination

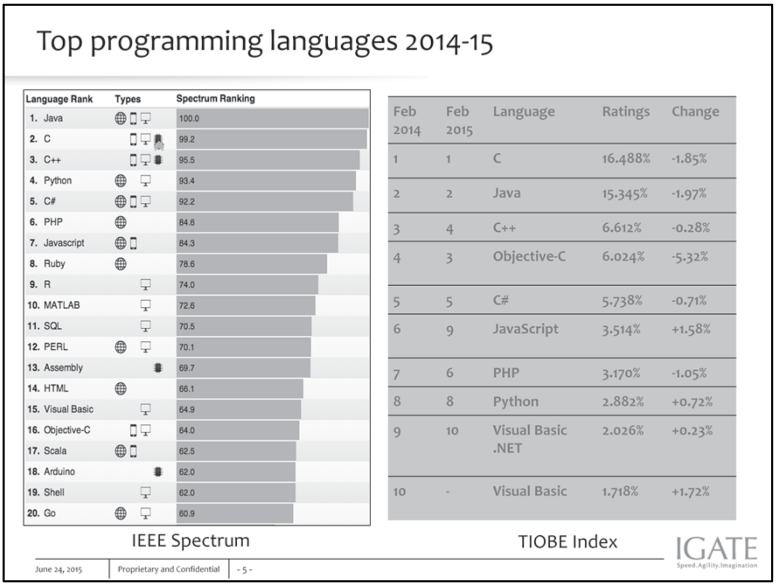


What is Python?

- Python is a
  - High level programming language
  - Interpreted
  - Interactive
  - Object Oriented
- Python development started in December 1989. Designed by and principal author Guido van Rossum
- Influences from other languages
  - ABC : Core syntax directly inherited
  - Bourne shell : Interactive Interpreter
  - Lisp & Haskell : Features such as list comprehensions, map functions
  - Perl : Regular expressions, shell script

June 24, 2015 | Proprietary and Confidential | - 4 -

Speed.Agility.Imagination



Top programming languages 2014-15

Language Rank	Types	Spectrum Ranking
1. Java	🌐 📱 🖥	100.0
2. C	📱 🖥 ▪	99.2
3. C++	📱 🖥 ▪	95.5
4. Python	🌐 🖥	93.4
5. C#	🌐 📱 🖥	92.2
6. PHP	🌐	84.6
7. Javascript	🌐 📱	84.3
8. Ruby	🌐	78.6
9. R	🖥	74.0
10. MATLAB	🖥	72.6
11. SQL	🖥	70.5
12. PERL	🌐 🖥	70.1
13. Assembly	▪	69.7
14. HTML	🌐	66.1
15. Visual Basic	🖥	64.9
16. Objective-C	📱 🖥	64.0
17. Scala	🌐 📱	62.5
18. Arduino	▪	62.0
19. Shell	🖥	62.0
20. Go	🌐 🖥	60.9

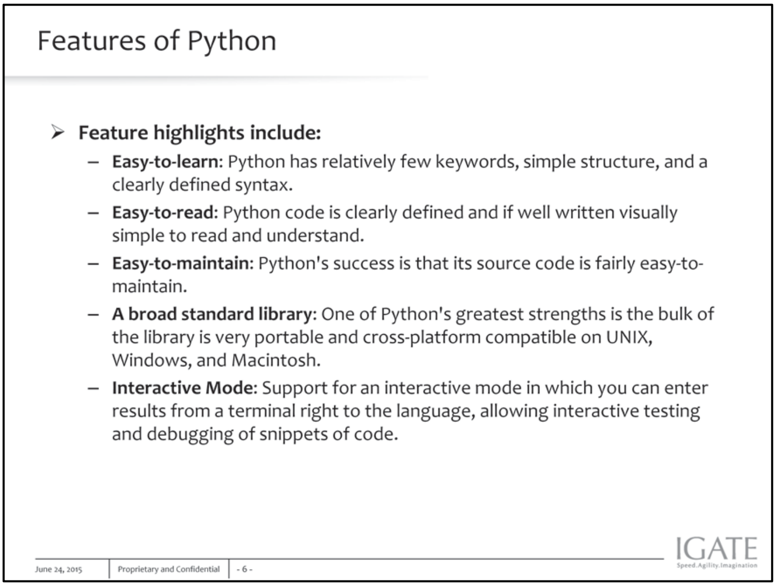
IEEE Spectrum

Feb 2014	Feb 2015	Language	Ratings	Change
1	1	C	16.488%	-1.85%
2	2	Java	15.345%	-1.97%
3	4	C++	6.612%	-0.28%
4	3	Objective-C	6.024%	-5.32%
5	5	C#	5.738%	-0.71%
6	9	JavaScript	3.514%	+1.58%
7	6	PHP	3.170%	-1.05%
8	8	Python	2.882%	+0.72%
9	10	Visual Basic .NET	2.026%	+0.23%
10	-	Visual Basic	1.718%	+1.72%

TIOBE Index

June 24, 2015 | Proprietary and Confidential | - 5 -

IGATE
Speed.Agility.Imagination

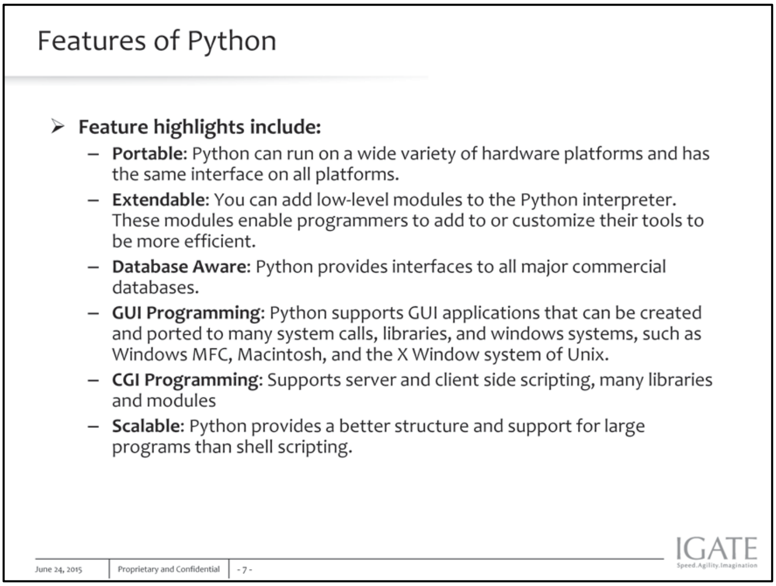


Features of Python

- **Feature highlights include:**
  - **Easy-to-learn:** Python has relatively few keywords, simple structure, and a clearly defined syntax.
  - **Easy-to-read:** Python code is clearly defined and if well written visually simple to read and understand.
  - **Easy-to-maintain:** Python's success is that its source code is fairly easy-to-maintain.
  - **A broad standard library:** One of Python's greatest strengths is the bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
  - **Interactive Mode:** Support for an interactive mode in which you can enter results from a terminal right to the language, allowing interactive testing and debugging of snippets of code.

June 24, 2015 | Proprietary and Confidential | - 6 -

IGATE
Speed.Agility.Imagination

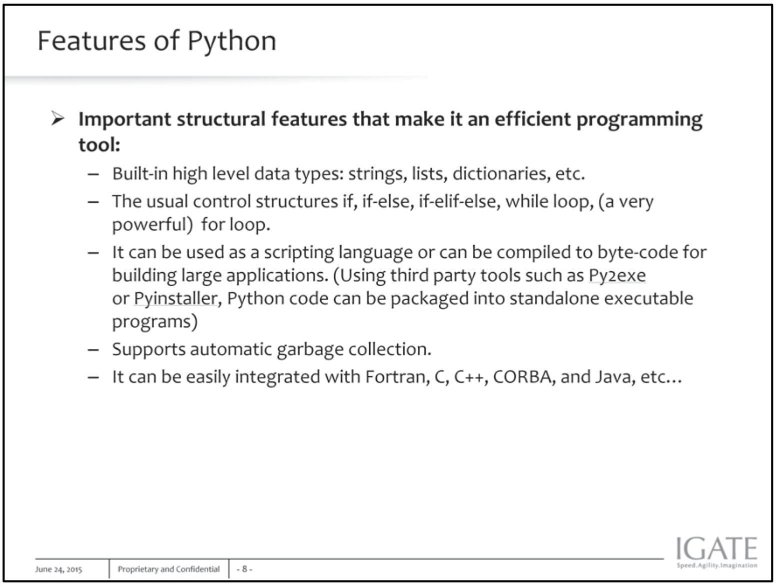


Features of Python

- **Feature highlights include:**
  - **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
  - **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
  - **Database Aware**: Python provides interfaces to all major commercial databases.
  - **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
  - **CGI Programming**: Supports server and client side scripting, many libraries and modules
  - **Scalable**: Python provides a better structure and support for large programs than shell scripting.

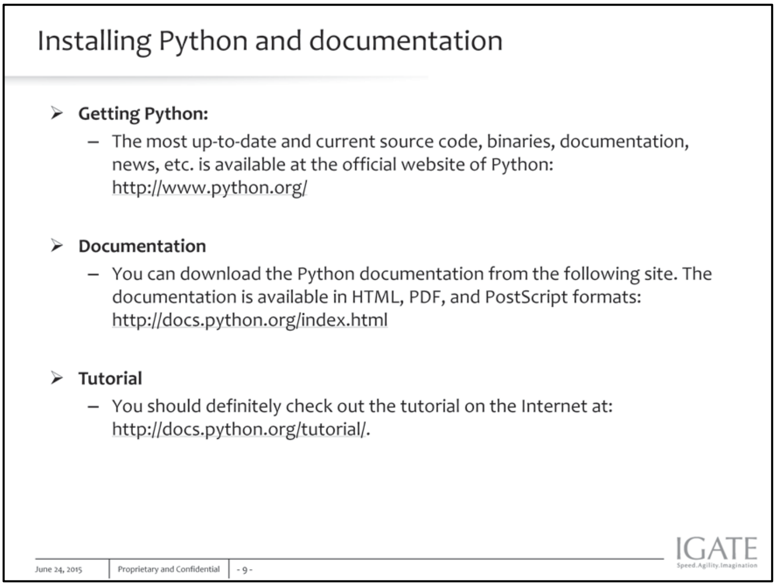
June 24, 2015 | Proprietary and Confidential | - 7 -

IGATE
Speed.Agility.Imagination



Features of Python

- **Important structural features that make it an efficient programming tool:**
  - Built-in high level data types: strings, lists, dictionaries, etc.
  - The usual control structures if, if-else, if-elif-else, while loop, (a very powerful) for loop.
  - It can be used as a scripting language or can be compiled to byte-code for building large applications. (Using third party tools such as Py2exe or Pyinstaller, Python code can be packaged into standalone executable programs)
  - Supports automatic garbage collection.
  - It can be easily integrated with Fortran, C, C++, CORBA, and Java, etc…



Installing Python and documentation

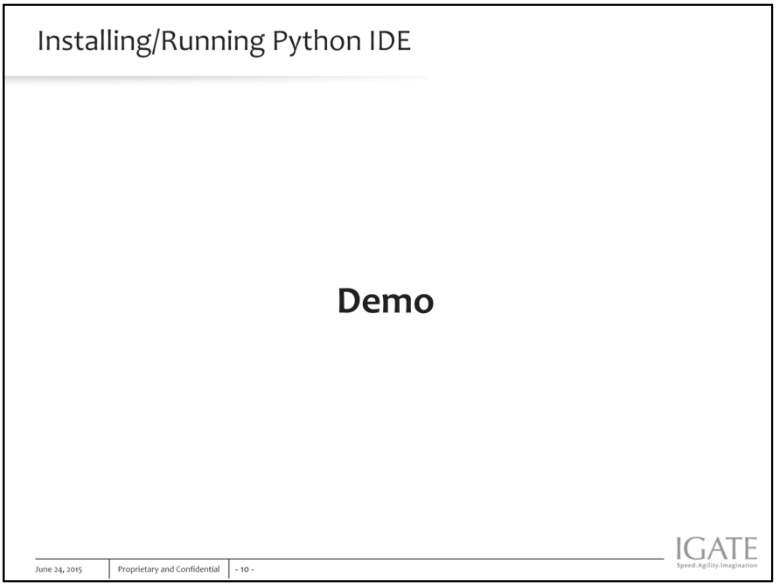
- **Getting Python:**
  - The most up-to-date and current source code, binaries, documentation, news, etc. is available at the official website of Python: http://www.python.org/

- **Documentation**
  - You can download the Python documentation from the following site. The documentation is available in HTML, PDF, and PostScript formats: http://docs.python.org/index.html

- **Tutorial**
  - You should definitely check out the tutorial on the Internet at: http://docs.python.org/tutorial/.

June 24, 2015 | Proprietary and Confidential | - 9 -

IGATE Speed.Agility.Imagination



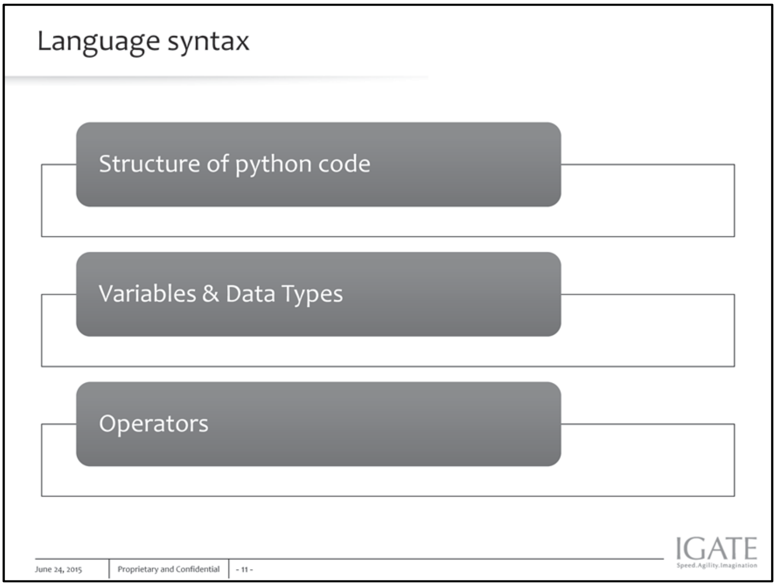
Installing/Running Python IDE

Demo

June 24, 2015 | Proprietary and Confidential | - 10 -

IGATE
Speed.Agility.Imagination

Installation from a shared directory onto the audience systems to be given
If already installation is done then demo of IDE to be done

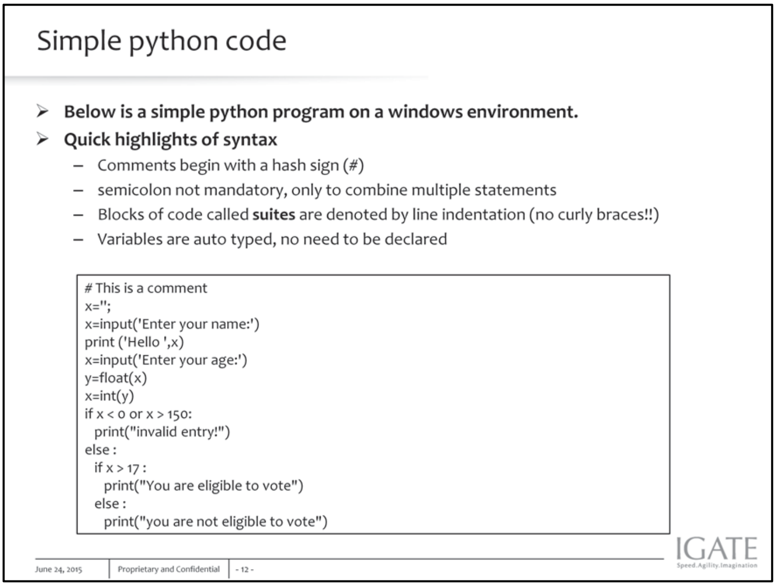


Language syntax

- Structure of python code
- Variables & Data Types
- Operators

June 24, 2015 | Proprietary and Confidential | - 11 -

IGATE
Speed.Agility.Imagination

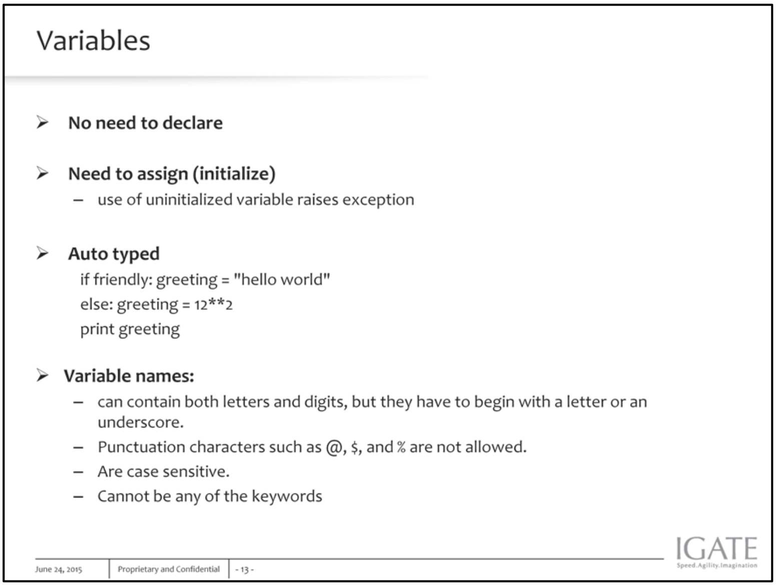


Simple python code

- **Below is a simple python program on a windows environment.**
- **Quick highlights of syntax**
  - Comments begin with a hash sign (#)
  - semicolon not mandatory, only to combine multiple statements
  - Blocks of code called **suites** are denoted by line indentation (no curly braces!!)
  - Variables are auto typed, no need to be declared

```
# This is a comment
x='';
x=input('Enter your name:')
print ('Hello ',x)
x=input('Enter your age:')
y=float(x)
x=int(y)
if x < 0 or x > 150:
  print("invalid entry!")
else :
  if x > 17 :
    print("You are eligible to vote")
  else :
    print("you are not eligible to vote")
```

June 24, 2015 | Proprietary and Confidential | - 12 -



Variables

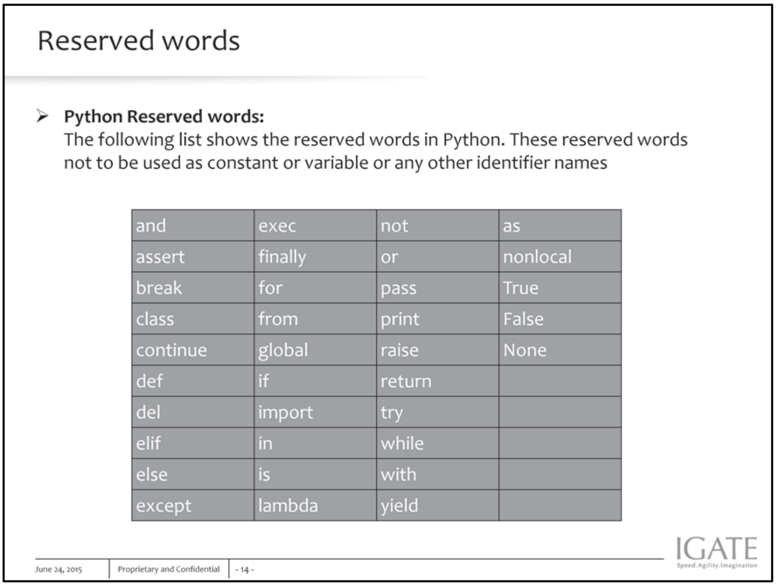
- **No need to declare**
- **Need to assign (initialize)**
  - use of uninitialized variable raises exception
- **Auto typed**

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- **Variable names:**
  - can contain both letters and digits, but they have to begin with a letter or an underscore.
  - Punctuation characters such as @, $, and % are not allowed.
  - Are case sensitive.
  - Cannot be any of the keywords

June 24, 2015 | Proprietary and Confidential | - 13 -

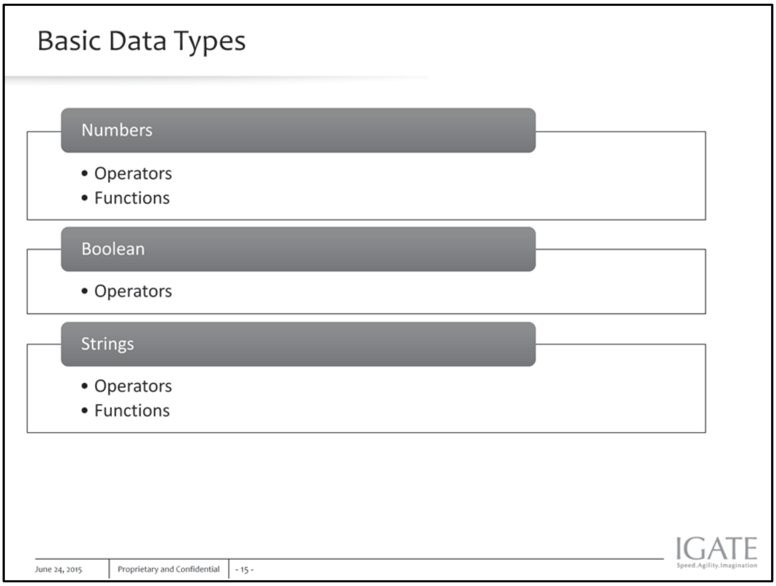
IGATE
Speed.Agility.Imagination



Reserved words

- **Python Reserved words:**
  The following list shows the reserved words in Python. These reserved words not to be used as constant or variable or any other identifier names

and	exec	not	as
assert	finally	or	nonlocal
break	for	pass	True
class	from	print	False
continue	global	raise	None
def	if	return	
del	import	try	
elif	in	while	
else	is	with	
except	lambda	yield	



Basic Data Types

Numbers

- Operators
- Functions

Boolean

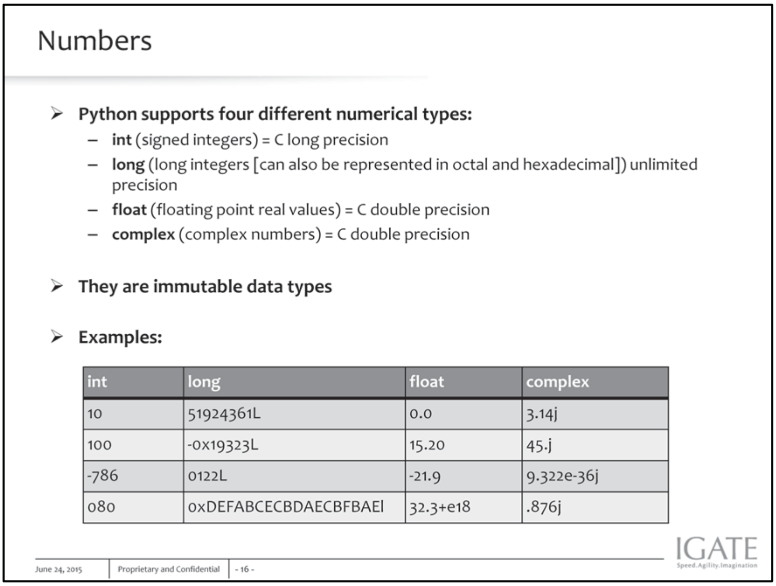
- Operators

Strings

- Operators
- Functions

June 24, 2015 | Proprietary and Confidential | - 15 -

IGATE
Speed.Agility.Imagination



Numbers

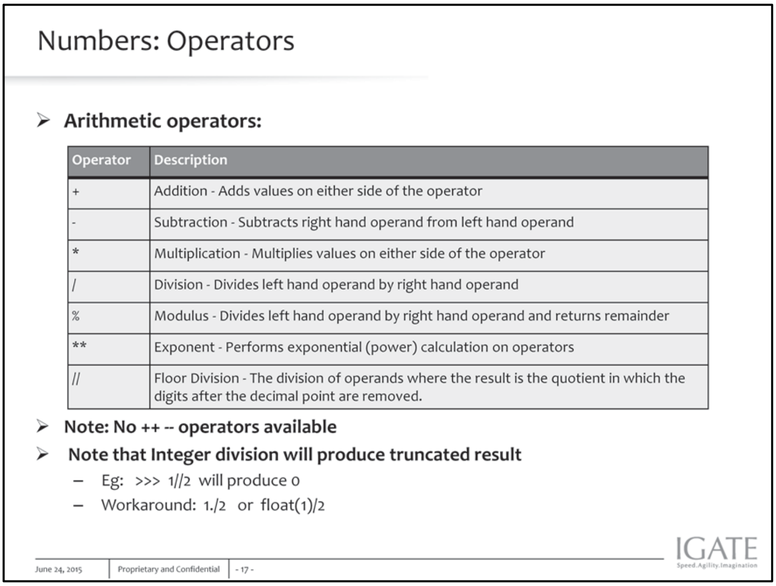
- **Python supports four different numerical types:**
  - **int** (signed integers) = C long precision
  - **long** (long integers [can also be represented in octal and hexadecimal]) unlimited precision
  - **float** (floating point real values) = C double precision
  - **complex** (complex numbers) = C double precision
- **They are immutable data types**
- **Examples:**

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j

June 24, 2015 | Proprietary and Confidential | - 16 -

IGATE
Speed.Agility.Imagination

Immutable means that changing the value of a number data type results in a newly allocated object



Numbers: Operators

- **Arithmetic operators:**

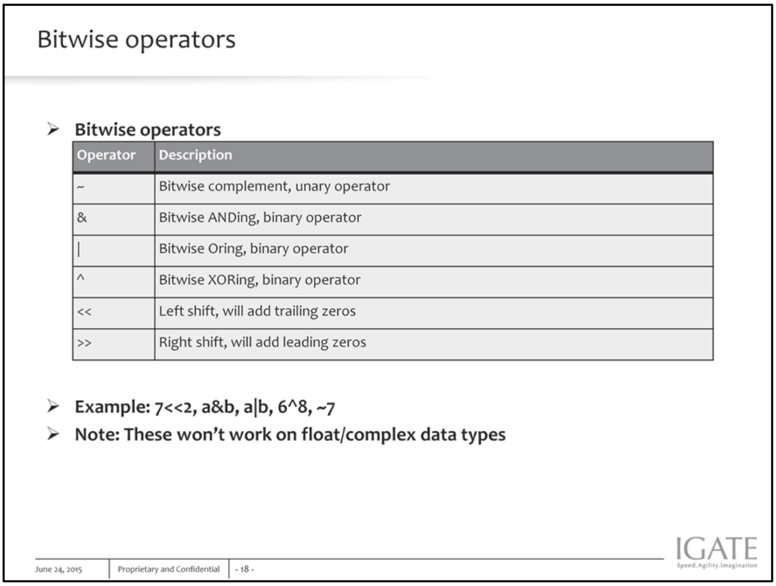
Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

- **Note: No ++ -- operators available**
- **Note that Integer division will produce truncated result**
  - Eg: >>> 1//2 will produce 0
  - Workaround: 1./2 or float(1)/2

June 24, 2015 | Proprietary and Confidential | - 17 -

IGATE
Speed.Agility.Imagination

Immutable means that changing the value of a number data type results in a newly allocated object



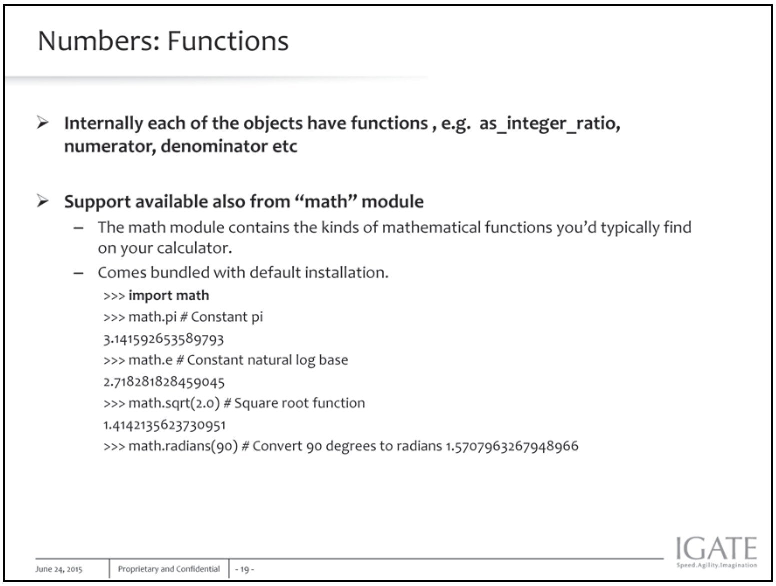
Bitwise operators

- **Bitwise operators**

Operator	Description
~	Bitwise complement, unary operator
&	Bitwise ANDing, binary operator
|	Bitwise Oring, binary operator
^	Bitwise XORing, binary operator
<<	Left shift, will add trailing zeros
>>	Right shift, will add leading zeros

- **Example: 7<<2, a&b, a|b, 6^8, ~7**
- **Note: These won't work on float/complex data types**

June 24, 2015 | Proprietary and Confidential | - 18 -



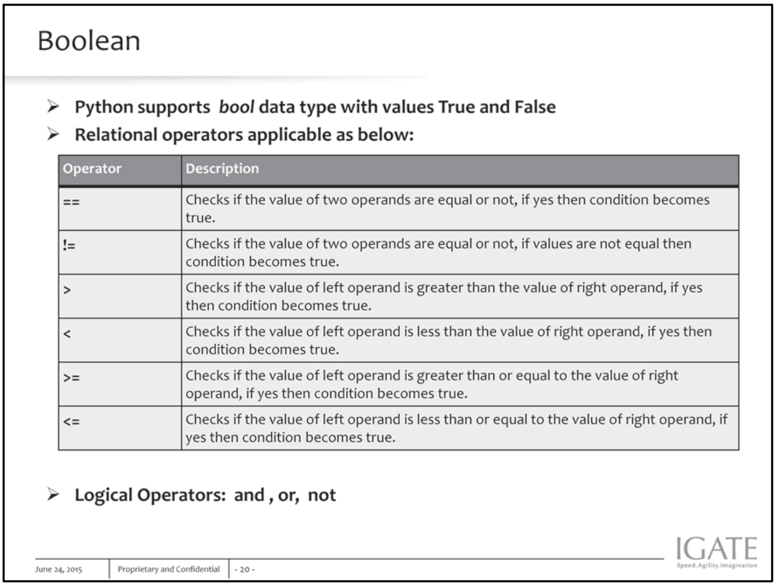
Numbers: Functions

- **Internally each of the objects have functions , e.g. as_integer_ratio, numerator, denominator etc**

- **Support available also from "math" module**
  - The math module contains the kinds of mathematical functions you'd typically find on your calculator.
  - Comes bundled with default installation.

```
>>> import math
>>> math.pi # Constant pi
3.141592653589793
>>> math.e # Constant natural log base
2.718281828459045
>>> math.sqrt(2.0) # Square root function
1.4142135623730951
>>> math.radians(90) # Convert 90 degrees to radians 1.5707963267948966
```

June 24, 2015 | Proprietary and Confidential | - 19 -



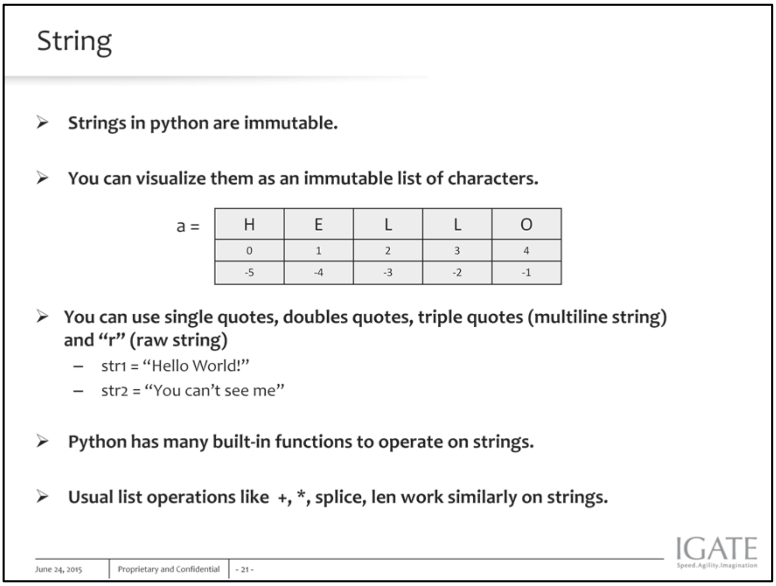
Boolean

- Python supports *bool* data type with values True and False
- Relational operators applicable as below:

Operator	Description
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

- Logical Operators: and , or, not

IGATE
Speed.Agility.Imagination



String

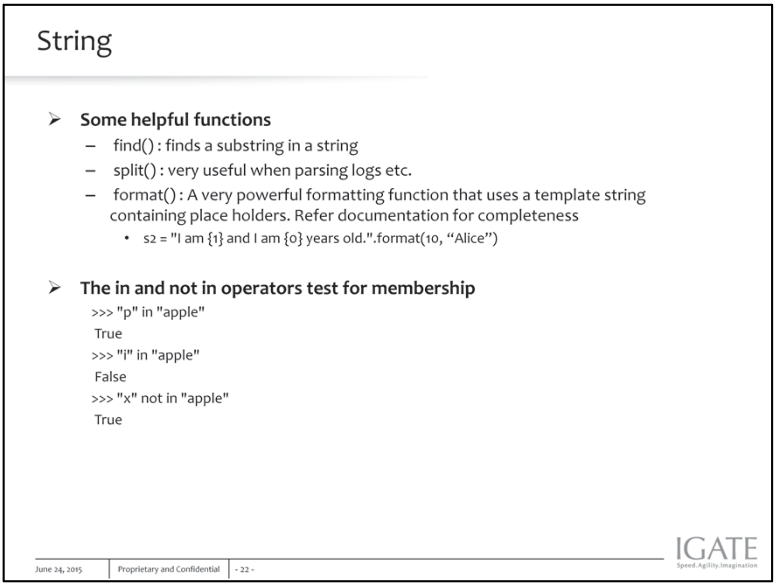
- Strings in python are immutable.
- You can visualize them as an immutable list of characters.

a =	H	E	L	L	O
	0	1	2	3	4
	-5	-4	-3	-2	-1

- You can use single quotes, doubles quotes, triple quotes (multiline string) and “r” (raw string)
  - str1 = “Hello World!”
  - str2 = “You can’t see me”
- Python has many built-in functions to operate on strings.
- Usual list operations like +, *, splice, len work similarly on strings.

June 24, 2015 | Proprietary and Confidential | - 21 -

IGATE
Speed.Agility.Imagination



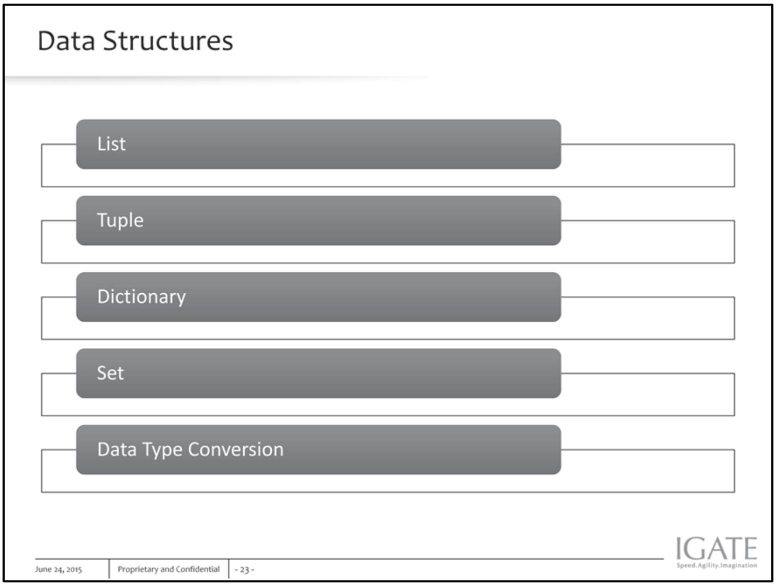
String

- **Some helpful functions**
  - find() : finds a substring in a string
  - split() : very useful when parsing logs etc.
  - format() : A very powerful formatting function that uses a template string containing place holders. Refer documentation for completeness
    - s2 = "I am {1} and I am {0} years old.".format(10, “Alice”)

- **The in and not in operators test for membership**

```
>>> "p" in "apple"
 True
>>> "i" in "apple"
 False
>>> "x" not in "apple"
 True
```

June 24, 2015 | Proprietary and Confidential | - 22 -

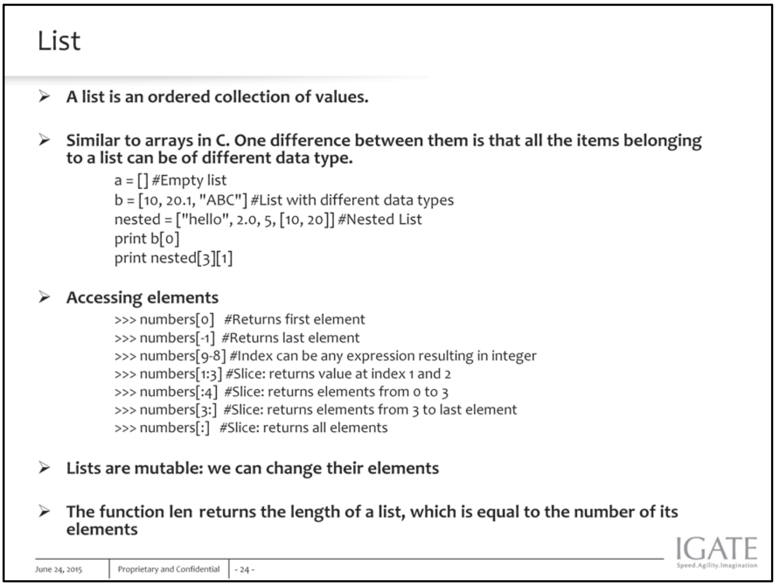


Data Structures

- List
- Tuple
- Dictionary
- Set
- Data Type Conversion

June 24, 2015 | Proprietary and Confidential | - 23 -

IGATE Speed.Agility.Imagination



List

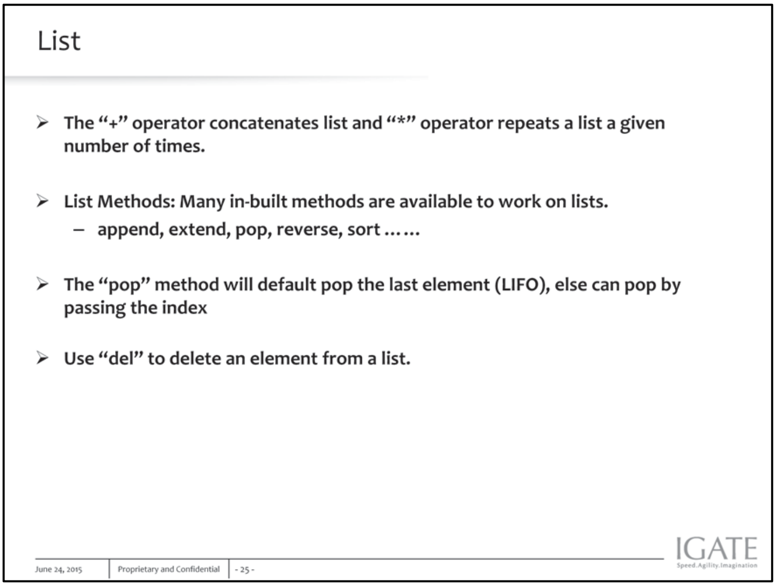
- A list is an ordered collection of values.
- Similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

```
a = [] #Empty list
b = [10, 20.1, "ABC"] #List with different data types
nested = ["hello", 2.0, 5, [10, 20]] #Nested List
print b[0]
print nested[3][1]
```

- Accessing elements

```
>>> numbers[0]   #Returns first element
>>> numbers[-1]  #Returns last element
>>> numbers[9-8] #Index can be any expression resulting in integer
>>> numbers[1:3] #Slice: returns value at index 1 and 2
>>> numbers[:4]  #Slice: returns elements from 0 to 3
>>> numbers[3:]  #Slice: returns elements from 3 to last element
>>> numbers[:]   #Slice: returns all elements
```

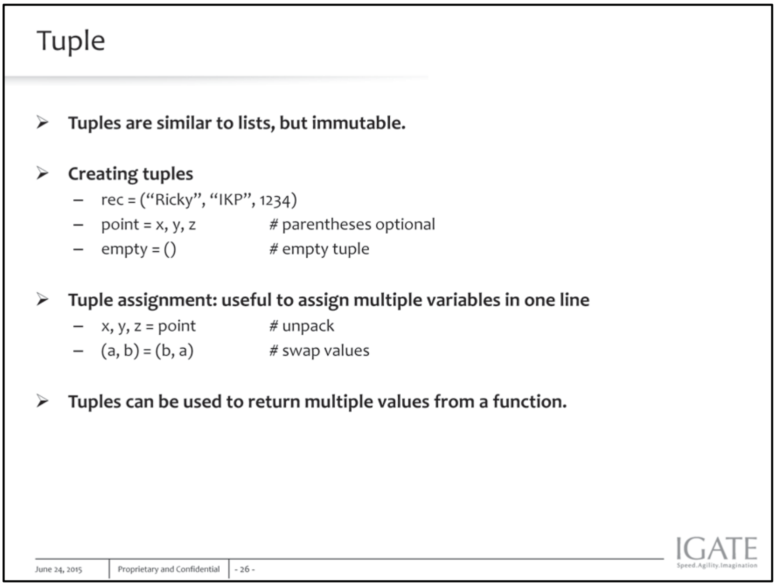
- Lists are mutable: we can change their elements
- The function len returns the length of a list, which is equal to the number of its elements



List

- The “+” operator concatenates list and “*” operator repeats a list a given number of times.
- List Methods: Many in-built methods are available to work on lists.
  - append, extend, pop, reverse, sort ……
- The “pop” method will default pop the last element (LIFO), else can pop by passing the index
- Use “del” to delete an element from a list.

June 24, 2015 | Proprietary and Confidential | - 25 -



Tuple

- **Tuples are similar to lists, but immutable.**

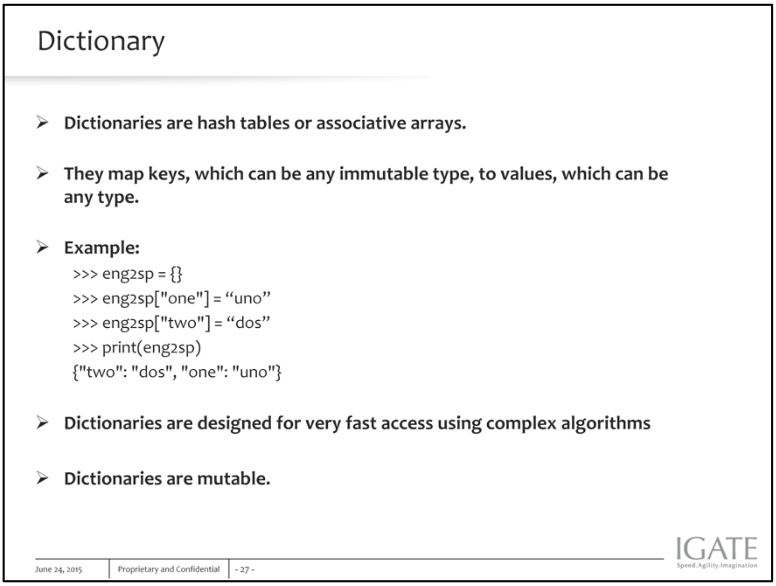
- **Creating tuples**
  - rec = (“Ricky”, “IKP”, 1234)
  - point = x, y, z # parentheses optional
  - empty = () # empty tuple

- **Tuple assignment: useful to assign multiple variables in one line**
  - x, y, z = point # unpack
  - (a, b) = (b, a) # swap values

- **Tuples can be used to return multiple values from a function.**

June 24, 2015 | Proprietary and Confidential | - 26 -

IGATE
Speed.Agility.Imagination



Dictionary

- Dictionaries are hash tables or associative arrays.

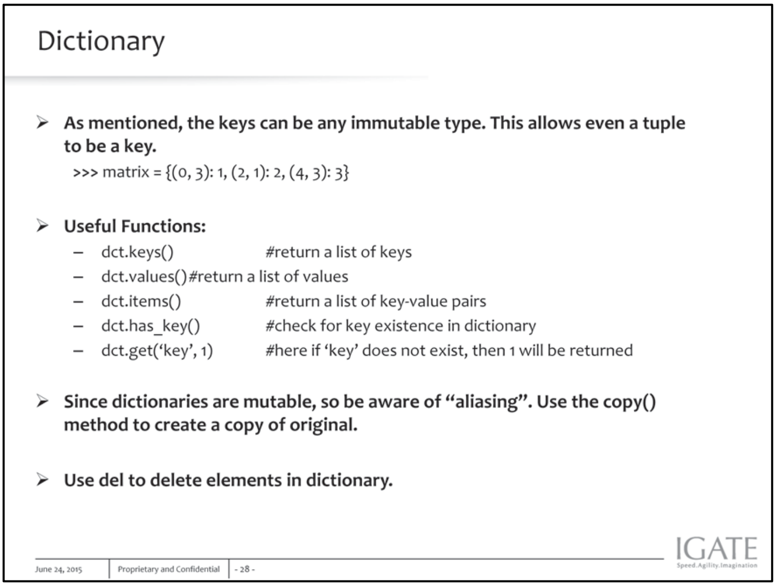
- They map keys, which can be any immutable type, to values, which can be any type.

- Example:

```
>>> eng2sp = {}
>>> eng2sp["one"] = “uno”
>>> eng2sp["two"] = “dos”
>>> print(eng2sp)
{"two": "dos", "one": "uno"}
```

- Dictionaries are designed for very fast access using complex algorithms

- Dictionaries are mutable.



Dictionary

- **As mentioned, the keys can be any immutable type. This allows even a tuple to be a key.**

  >>> matrix = {(0, 3): 1, (2, 1): 2, (4, 3): 3}

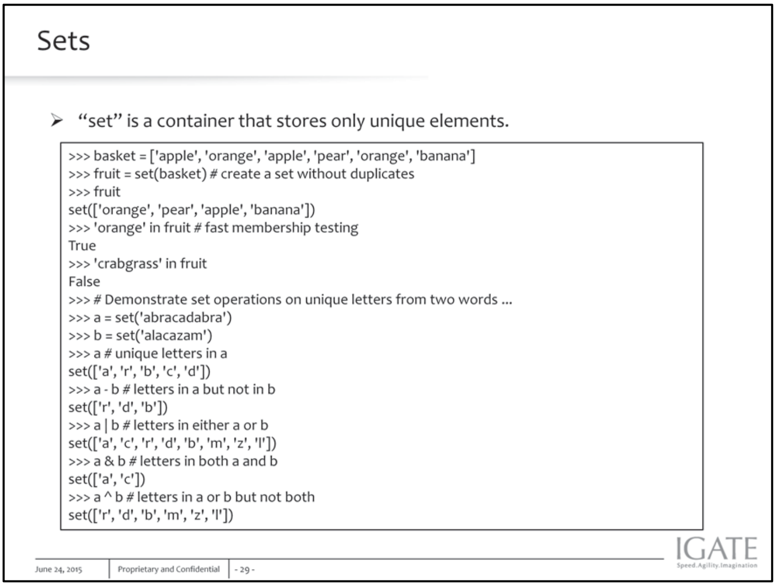
- **Useful Functions:**
  - dct.keys()   #return a list of keys
  - dct.values()#return a list of values
  - dct.items()   #return a list of key-value pairs
  - dct.has_key()   #check for key existence in dictionary
  - dct.get('key', 1)   #here if 'key' does not exist, then 1 will be returned

- **Since dictionaries are mutable, so be aware of "aliasing". Use the copy() method to create a copy of original.**

- **Use del to delete elements in dictionary.**

June 24, 2015 | Proprietary and Confidential | - 28 -

IGATE
Speed.Agility.Imagination

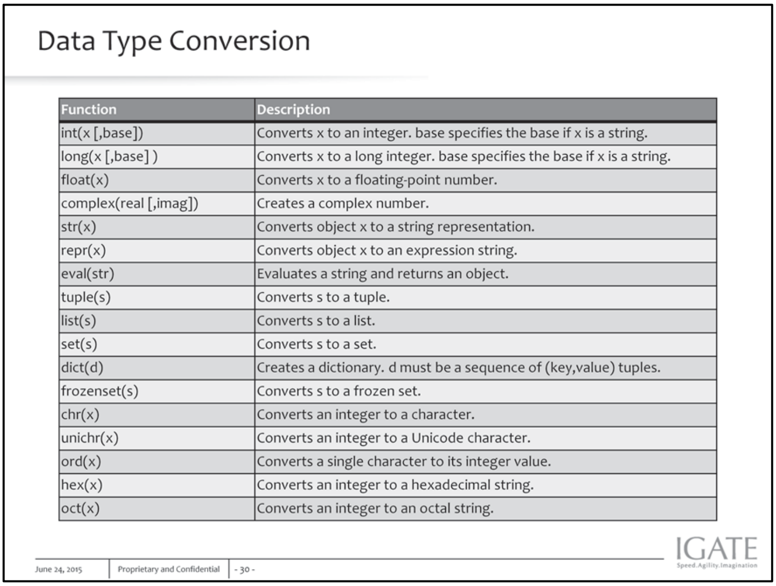


Sets

- “set” is a container that stores only unique elements.

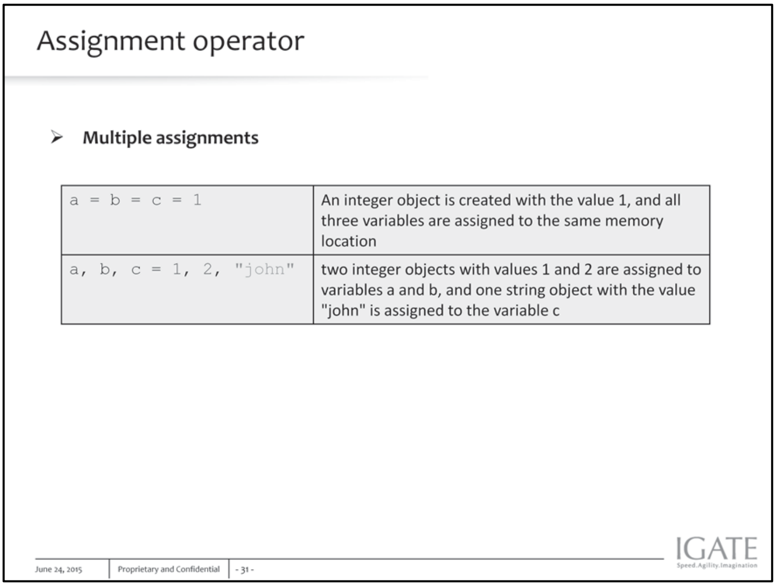
```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket) # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit # fast membership testing
True
>>> 'crabgrass' in fruit
False
>>> # Demonstrate set operations on unique letters from two words ...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b # letters in both a and b
set(['a', 'c'])
>>> a ^ b # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

June 24, 2015 | Proprietary and Confidential | - 29 -



Data Type Conversion

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.



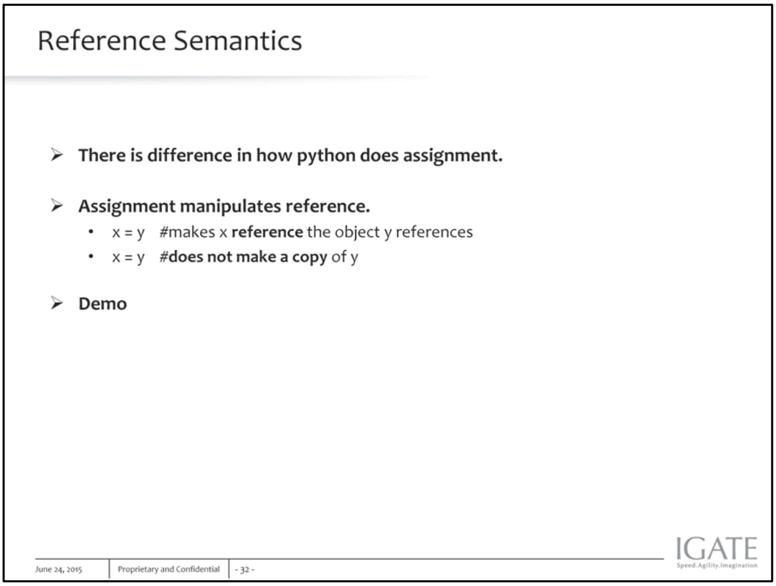
Assignment operator

- **Multiple assignments**

`a = b = c = 1`	An integer object is created with the value 1, and all three variables are assigned to the same memory location
`a, b, c = 1, 2, "john"`	two integer objects with values 1 and 2 are assigned to variables a and b, and one string object with the value "john" is assigned to the variable c

June 24, 2015 | Proprietary and Confidential | - 31 -

IGATE Speed.Agility.Imagination

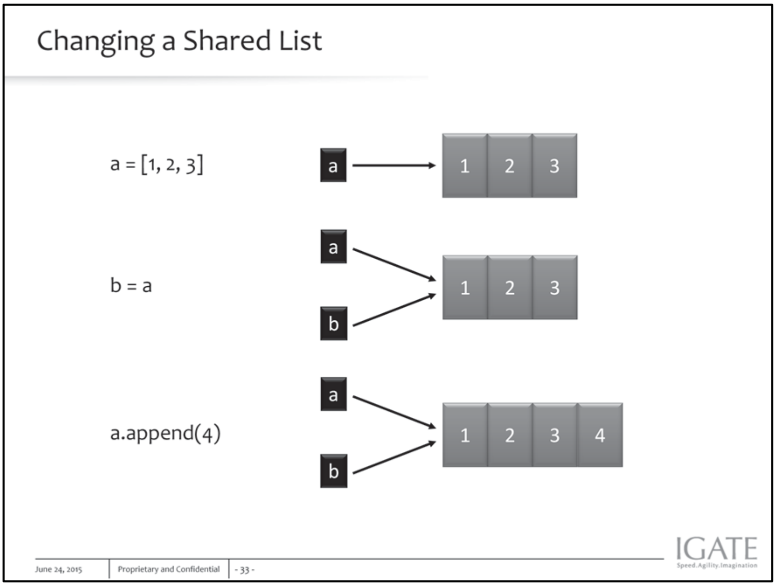


Reference Semantics

- **There is difference in how python does assignment.**
- **Assignment manipulates reference.**
  - x = y #makes x **reference** the object y references
  - x = y #**does not make a copy** of y
- **Demo**

June 24, 2015 | Proprietary and Confidential | - 32 -

IGATE
Speed.Agility.Imagination



Changing a Shared List

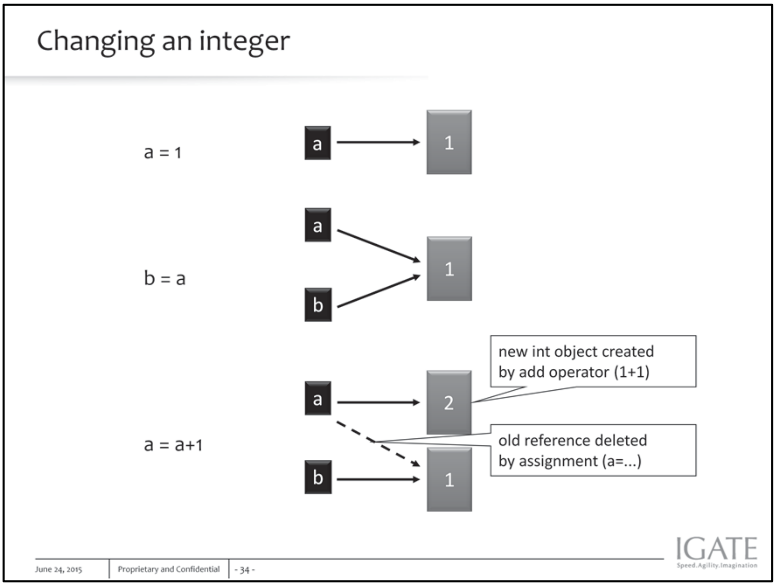
```
a = [1, 2, 3]
```

```
b = a
```

```
a.append(4)
```

June 24, 2015 | Proprietary and Confidential | - 33 -

IGATE
Speed.Agility.Imagination



Changing an integer

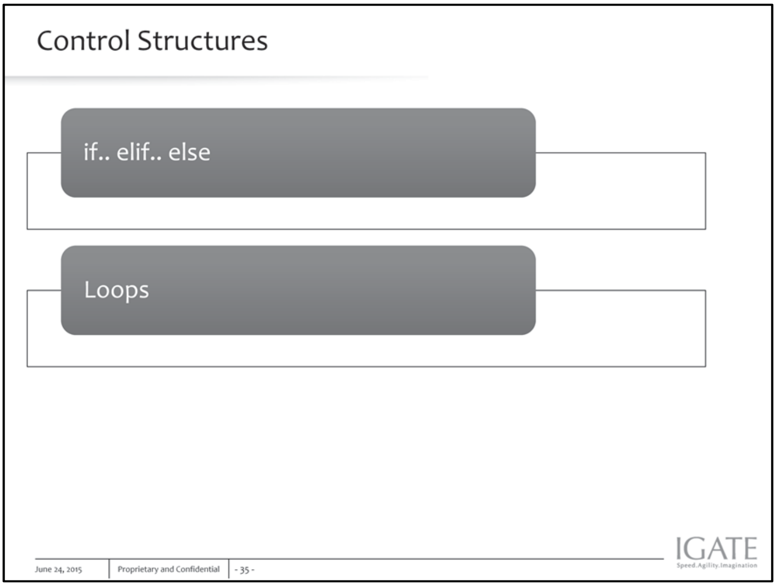
a = 1

b = a

a = a+1

new int object created by add operator (1+1)

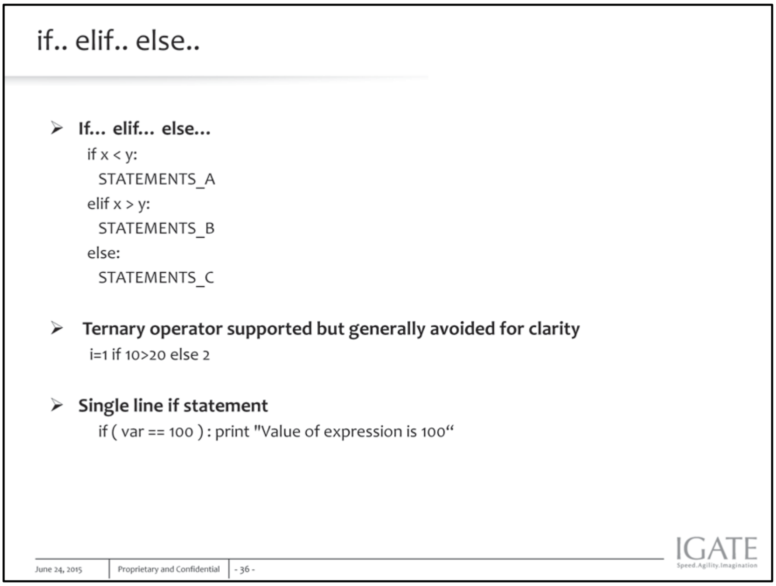
old reference deleted by assignment (a=...)



Control Structures

if.. elif.. else

Loops



if.. elif.. else..

- **If… elif… else…**

```
if x < y:
  STATEMENTS_A
elif x > y:
  STATEMENTS_B
else:
  STATEMENTS_C
```

- **Ternary operator supported but generally avoided for clarity**

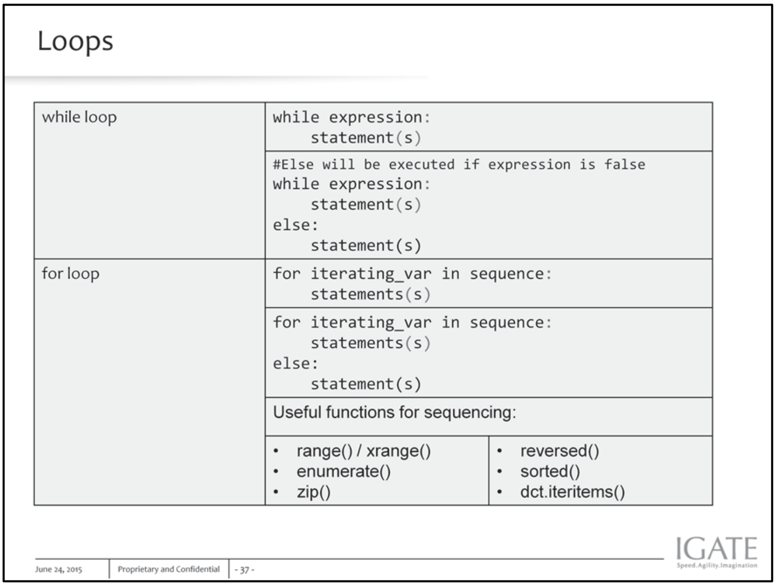
```
i=1 if 10>20 else 2
```

- **Single line if statement**

```
if ( var == 100 ) : print "Value of expression is 100“
```

June 24, 2015 | Proprietary and Confidential | - 36 -

IGATE
Speed.Agility.Imagination



Loops

while loop	while expression:
	    statement(s)	

	#Else will be executed if expression is false
	while expression:
	    statement(s)
	else:
	    statement(s)	

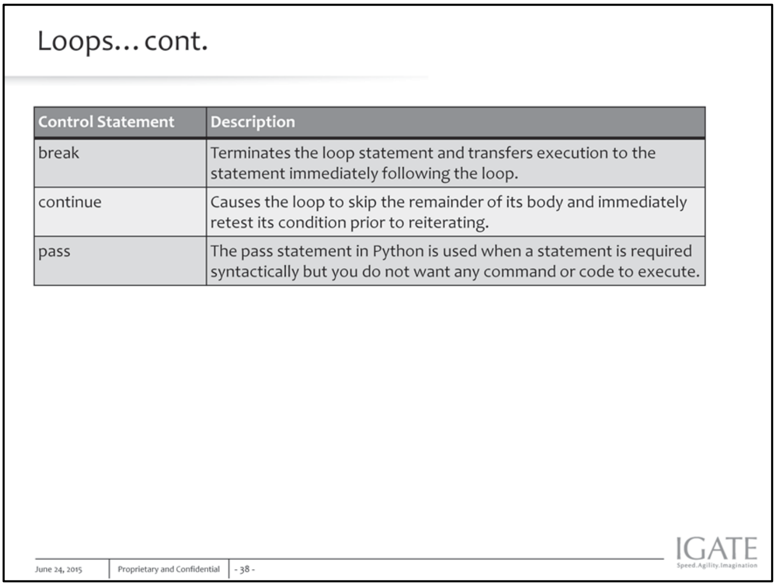
for loop	for iterating_var in sequence:
	    statements(s)	

	for iterating_var in sequence:
	    statements(s)
	else:
	    statement(s)	

	Useful functions for sequencing:	
	• range() / xrange() • enumerate() • zip()	• reversed() • sorted() • dct.iteritems()

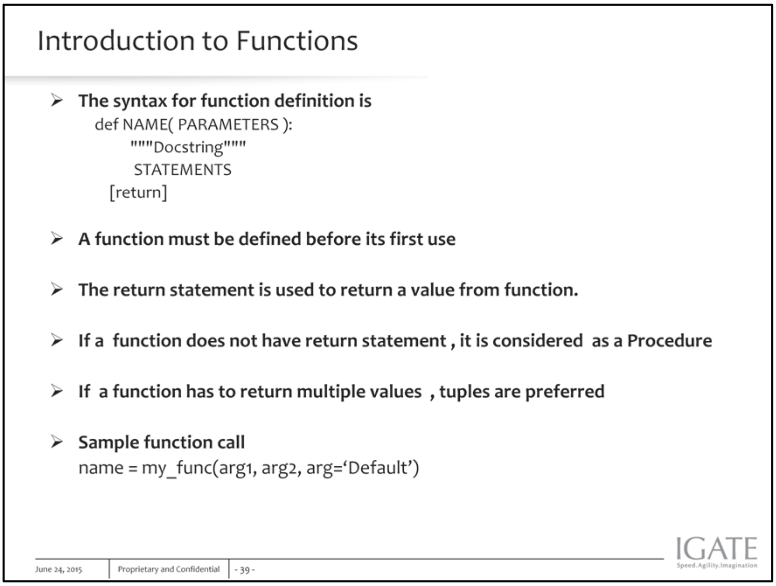
June 24, 2015 | Proprietary and Confidential | - 37 -

IGATE
Speed.Agility.Imagination



Loops… cont.

Control Statement	Description
break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.



Introduction to Functions

- **The syntax for function definition is**

```
  def NAME( PARAMETERS ):
      """Docstring"""
       STATEMENTS
    [return]
```

- **A function must be defined before its first use**
- **The return statement is used to return a value from function.**
- **If a function does not have return statement , it is considered as a Procedure**
- **If a function has to return multiple values , tuples are preferred**
- **Sample function call**
  name = my_func(arg1, arg2, arg='Default')

June 24, 2015 | Proprietary and Confidential | - 39 -

IGATE Speed.Agility.Imagination

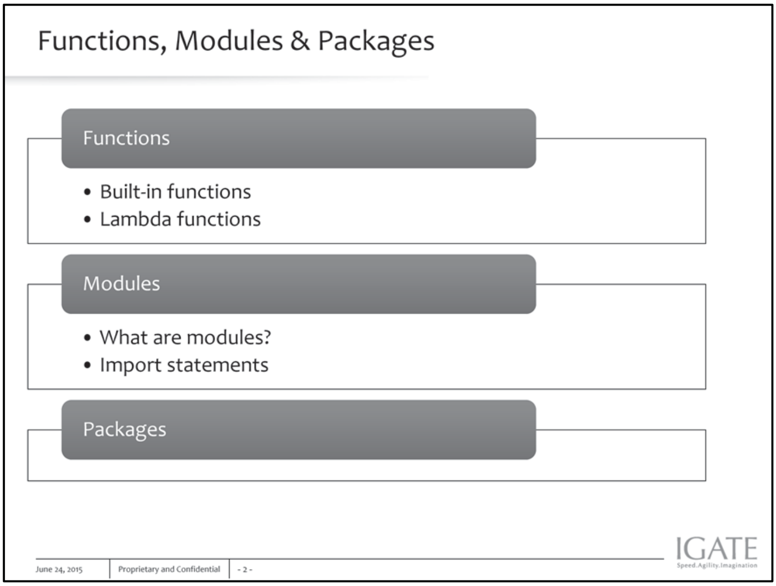
A function is one that returns a value. The opposite of
a fruitful function is void function — one that is not executed for its resulting value, but is
executed because it does something useful. (Languages like Java, C#, C and C++ use the term
"void function", other languages like Pascal call it a procedure.) Even though void functions
are not executed for their resulting value, Python always wants to return something. So if the
programmer doesn't arrange to return a value, Python will automatically return the value None

Python Training

A **basic** overview

IGATE

Speed.Agility.Imagination



Functions, Modules & Packages

Functions

- Built-in functions
- Lambda functions

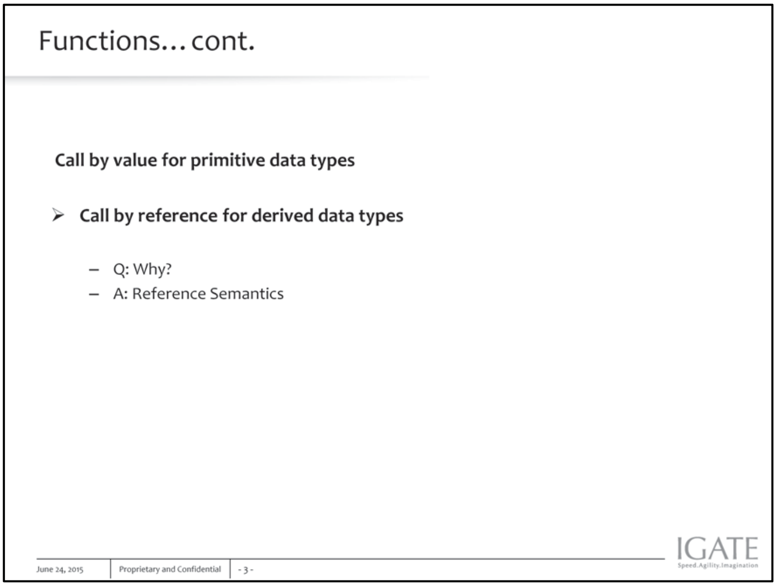
Modules

- What are modules?
- Import statements

Packages

June 24, 2015 | Proprietary and Confidential | - 2 -

IGATE
Speed.Agility.Imagination



Functions… cont.

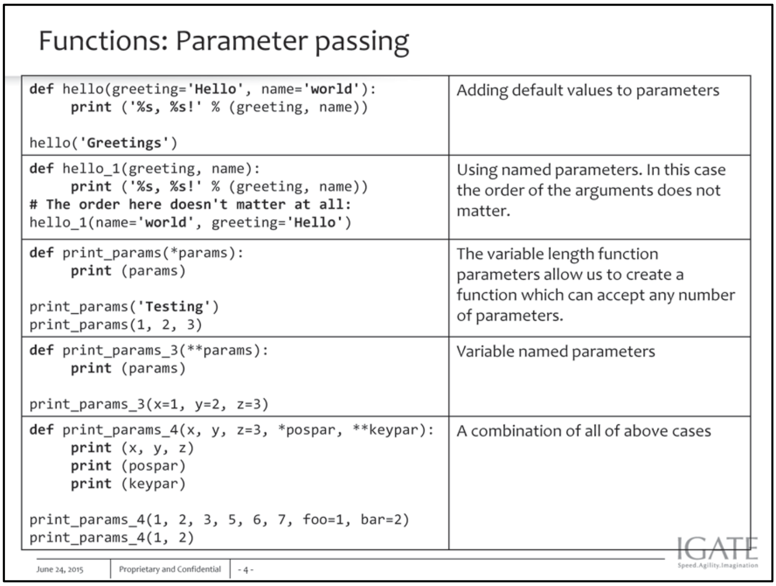
Call by value for primitive data types

- **Call by reference for derived data types**
  - Q: Why?
  - A: Reference Semantics

June 24, 2015 | Proprietary and Confidential | - 3 -

IGATE Speed.Agility.Imagination

A function is one that returns a value. The opposite of a fruitful function is void function — one that is not executed for its resulting value, but is executed because it does something useful. (Languages like Java, C#, C and C++ use the term "void function", other languages like Pascal call it a procedure.) Even though void functions are not executed for their resulting value, Python always wants to return something. So if the programmer doesn't arrange to return a value, Python will automatically return the value None



Functions: Parameter passing

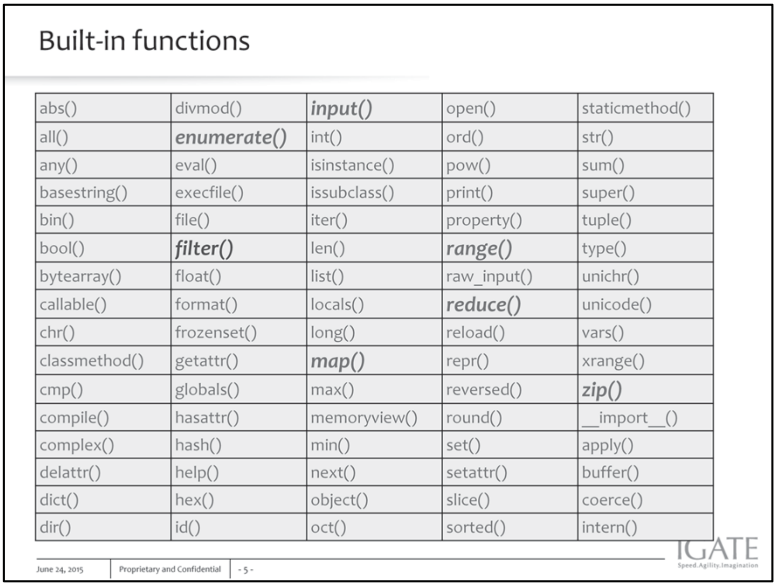
`def hello(greeting='Hello', name='world'):`
`     print ('%s, %s!' % (greeting, name))`
 	
`hello('Greetings')`
	Adding default values to parameters

`def hello_1(greeting, name):`
`     print ('%s, %s!' % (greeting, name))`
`# The order here doesn't matter at all:`
`hello_1(name='world', greeting='Hello')`
	Using named parameters. In this case the order of the arguments does not matter.

`def print_params(*params):`
`     print (params)`
 	
`print_params('Testing')`
`print_params(1, 2, 3)`
	The variable length function parameters allow us to create a function which can accept any number of parameters.

`def print_params_3(**params):`
`     print (params)`
 	
`print_params_3(x=1, y=2, z=3)`
	Variable named parameters

`def print_params_4(x, y, z=3, *pospar, **keypar):`
`     print (x, y, z)`
`     print (pospar)`
`     print (keypar)`
 	
`print_params_4(1, 2, 3, 5, 6, 7, foo=1, bar=2)`
`print_params_4(1, 2)`
	A combination of all of above cases

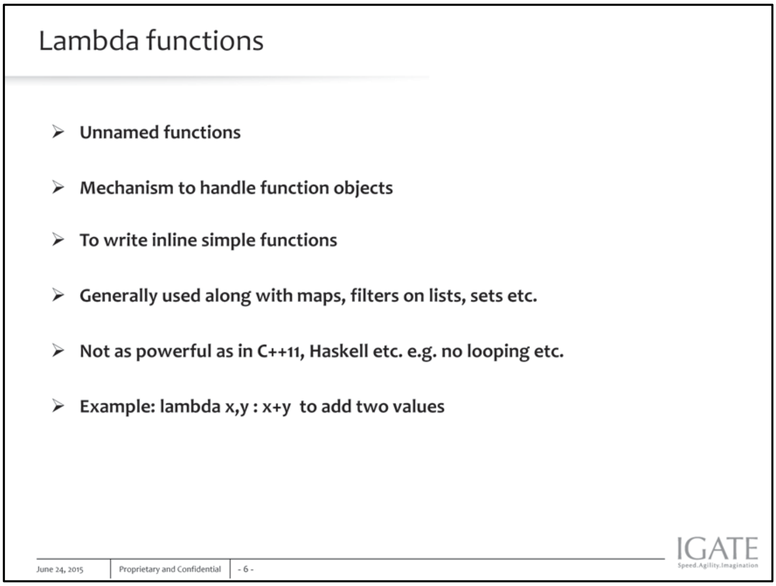


Built-in functions

abs()	divmod()	***input()***	open()	staticmethod()
all()	***enumerate()***	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	***filter()***	len()	***range()***	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	***reduce()***	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	***map()***	repr()	xrange()
cmp()	globals()	max()	reversed()	***zip()***
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	apply()
delattr()	help()	next()	setattr()	buffer()
dict()	hex()	object()	slice()	coerce()
dir()	id()	oct()	sorted()	intern()

June 24, 2015 | Proprietary and Confidential | - 5 -

IGATE
Speed.Agility.Imagination

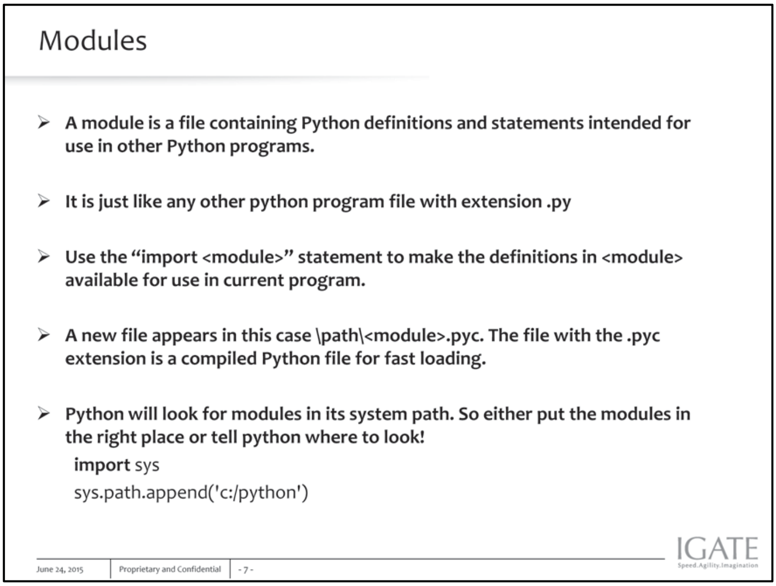


Lambda functions

- Unnamed functions
- Mechanism to handle function objects
- To write inline simple functions
- Generally used along with maps, filters on lists, sets etc.
- Not as powerful as in C++11, Haskell etc. e.g. no looping etc.
- Example: lambda x,y : x+y to add two values

June 24, 2015 | Proprietary and Confidential | - 6 -

IGATE
Speed.Agility.Imagination



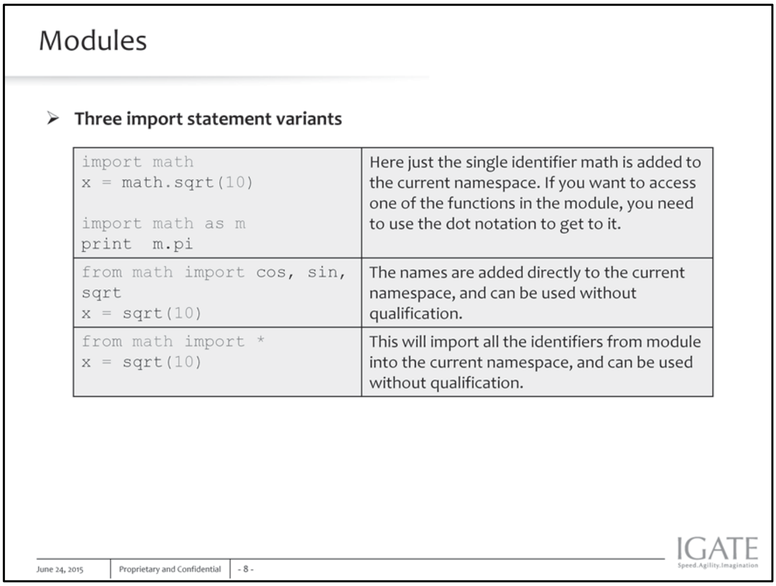
Modules

- A module is a file containing Python definitions and statements intended for use in other Python programs.
- It is just like any other python program file with extension .py
- Use the “import <module>” statement to make the definitions in <module> available for use in current program.
- A new file appears in this case \path\<module>.pyc. The file with the .pyc extension is a compiled Python file for fast loading.
- Python will look for modules in its system path. So either put the modules in the right place or tell python where to look!

```
import sys
sys.path.append('c:/python')
```

June 24, 2015 | Proprietary and Confidential | - 7 -

IGATE Speed.Agility.Imagination



Modules

- **Three import statement variants**

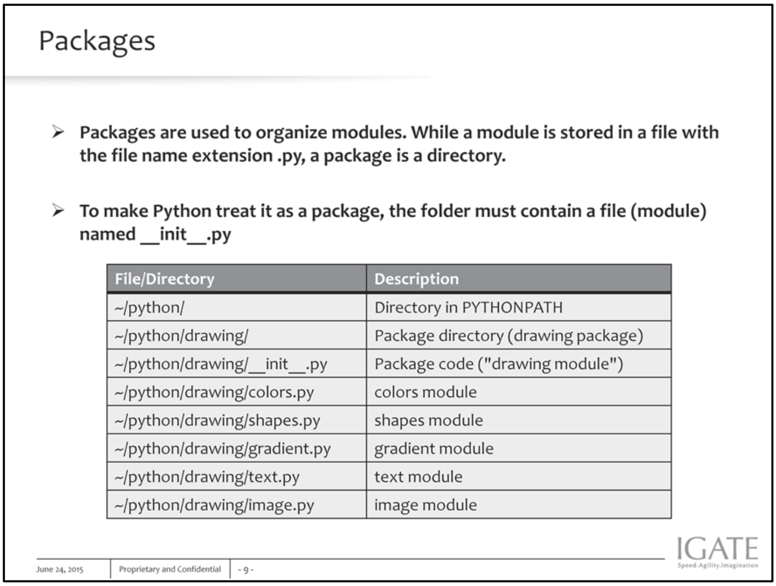
`import math`
`x = math.sqrt(10)`
 	
`import math as m`
`print  m.pi`
	Here just the single identifier math is added to the current namespace. If you want to access one of the functions in the module, you need to use the dot notation to get to it.

`from math import cos, sin,`
`sqrt`
`x = sqrt(10)`
	The names are added directly to the current namespace, and can be used without qualification.

`from math import *`
`x = sqrt(10)`
	This will import all the identifiers from module into the current namespace, and can be used without qualification.

June 24, 2015 | Proprietary and Confidential | - 8 -

IGATE
Speed.Agility.Imagination



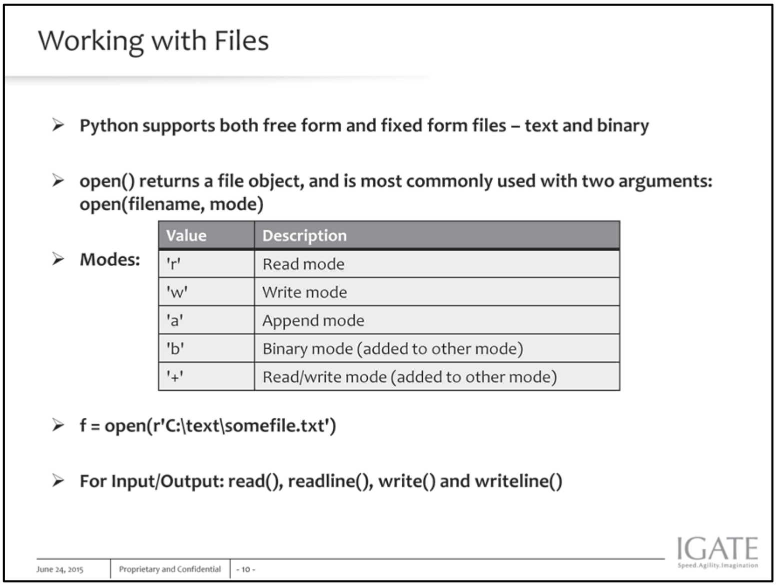
Packages

- Packages are used to organize modules. While a module is stored in a file with the file name extension .py, a package is a directory.
- To make Python treat it as a package, the folder must contain a file (module) named __init__.py

File/Directory	Description
~/python/	Directory in PYTHONPATH
~/python/drawing/	Package directory (drawing package)
~/python/drawing/__init__.py	Package code ("drawing module")
~/python/drawing/colors.py	colors module
~/python/drawing/shapes.py	shapes module
~/python/drawing/gradient.py	gradient module
~/python/drawing/text.py	text module
~/python/drawing/image.py	image module

June 24, 2015 | Proprietary and Confidential | - 9 -

IGATE
Speed.Agility.Imagination



Working with Files

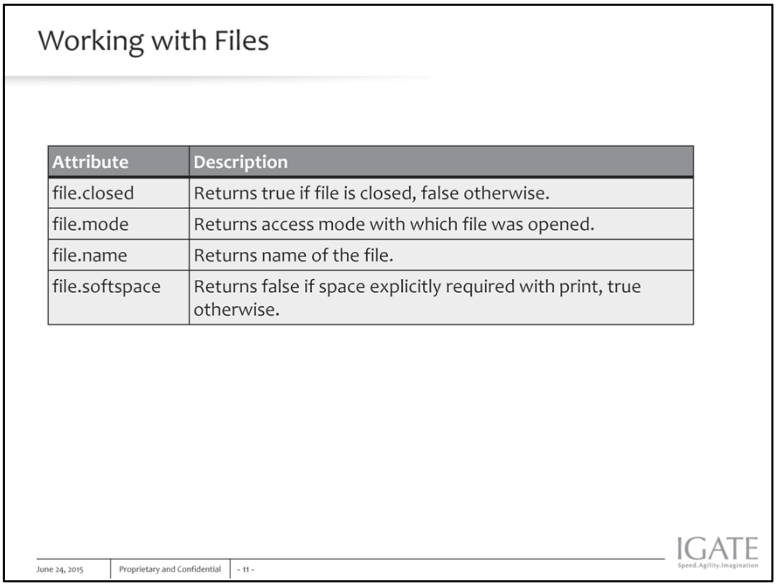
- Python supports both free form and fixed form files – text and binary
- open() returns a file object, and is most commonly used with two arguments: open(filename, mode)
- Modes:

Value	Description
'r'	Read mode
'w'	Write mode
'a'	Append mode
'b'	Binary mode (added to other mode)
'+'	Read/write mode (added to other mode)

- f = open(r'C:\text\somefile.txt')
- For Input/Output: read(), readline(), write() and writeline()

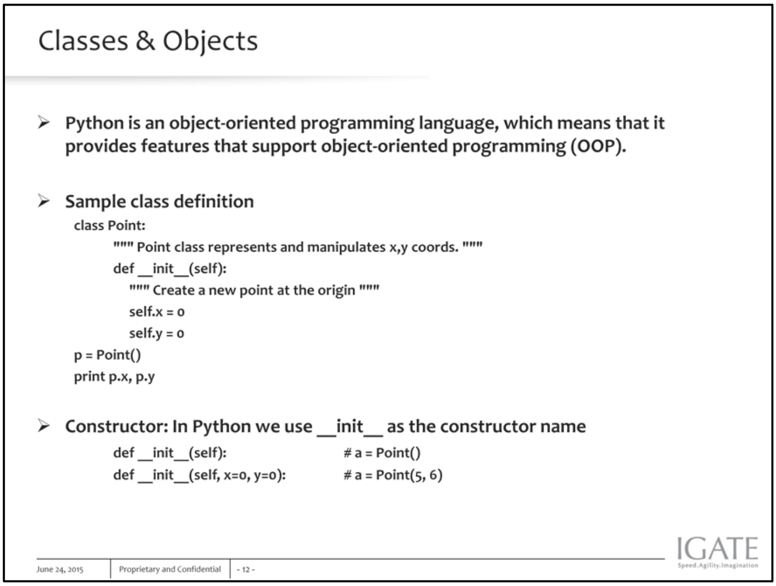
June 24, 2015 | Proprietary and Confidential | - 10 -

IGATE Speed.Agility.Imagination



Working with Files

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.



Classes & Objects

- Python is an object-oriented programming language, which means that it provides features that support object-oriented programming (OOP).

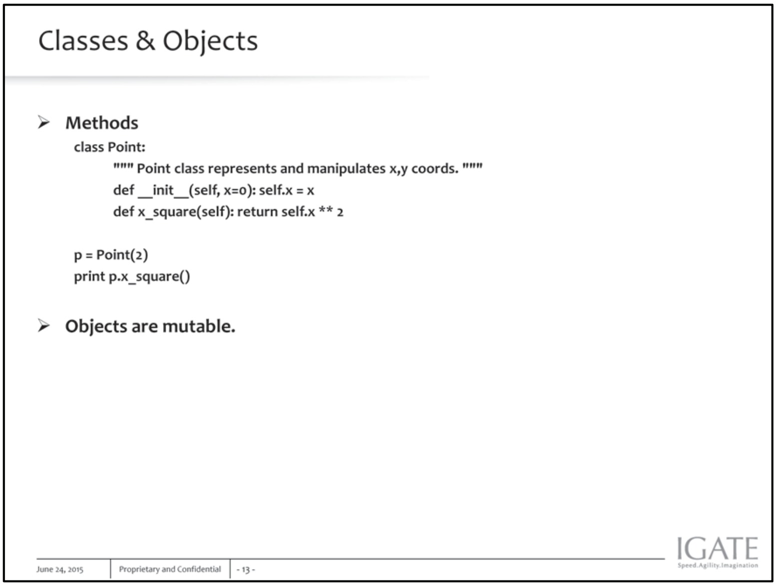
- Sample class definition

```
class Point:
        """ Point class represents and manipulates x,y coords. """
        def __init__(self):
           """ Create a new point at the origin """
           self.x = 0
           self.y = 0
p = Point()
print p.x, p.y
```

- Constructor: In Python we use __init__ as the constructor name

```
        def __init__(self):                  # a = Point()
        def __init__(self, x=0, y=0):        # a = Point(5, 6)
```

June 24, 2015 | Proprietary and Confidential | - 12 -



Classes & Objects

- **Methods**

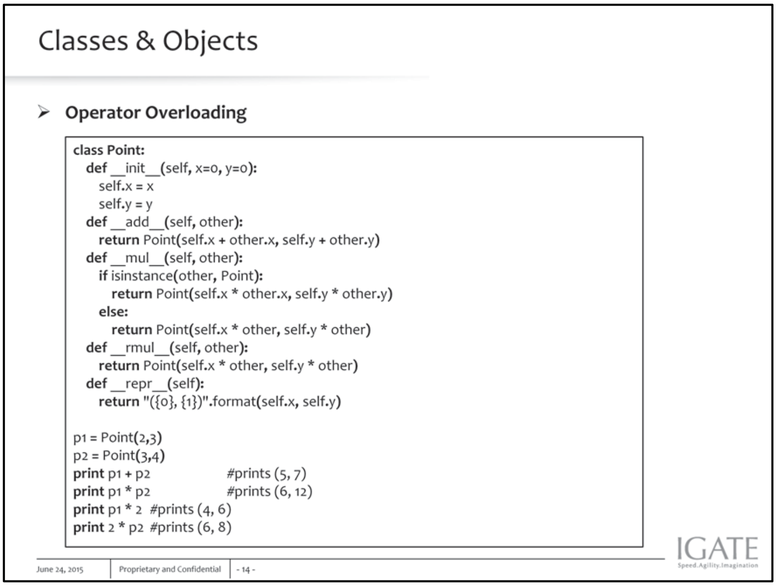
```
class Point:
        """ Point class represents and manipulates x,y coords. """
        def __init__(self, x=0): self.x = x
        def x_square(self): return self.x ** 2

p = Point(2)
print p.x_square()
```

- **Objects are mutable.**

June 24, 2015 | Proprietary and Confidential | - 13 -

IGATE
Speed.Agility.Imagination



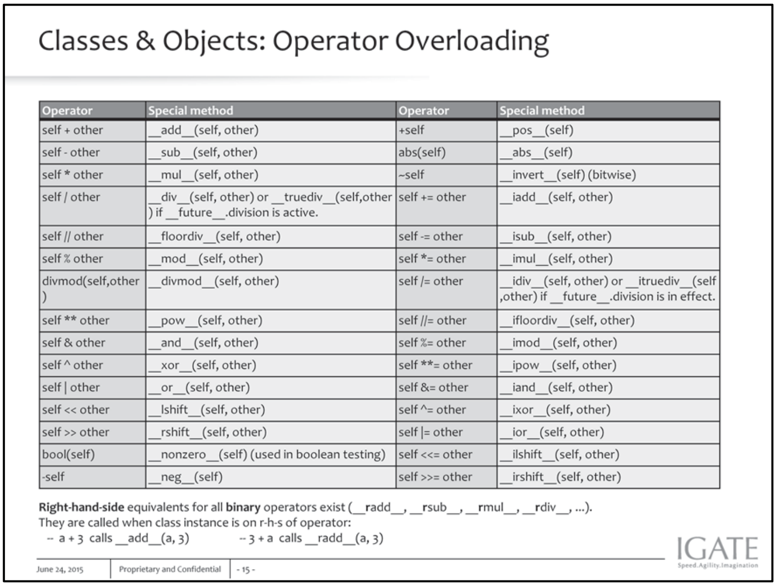
Classes & Objects

- **Operator Overloading**

```
class Point:
  def __init__(self, x=0, y=0):
    self.x = x
    self.y = y
  def __add__(self, other):
    return Point(self.x + other.x, self.y + other.y)
  def __mul__(self, other):
    if isinstance(other, Point):
      return Point(self.x * other.x, self.y * other.y)
    else:
      return Point(self.x * other, self.y * other)
  def __rmul__(self, other):
    return Point(self.x * other, self.y * other)
  def __repr__(self):
    return "({0}, {1})".format(self.x, self.y)

p1 = Point(2,3)
p2 = Point(3,4)
print p1 + p2              #prints (5, 7)
print p1 * p2              #prints (6, 12)
print p1 * 2  #prints (4, 6)
print 2 * p2  #prints (6, 8)
```

June 24, 2015 | Proprietary and Confidential | - 14 -

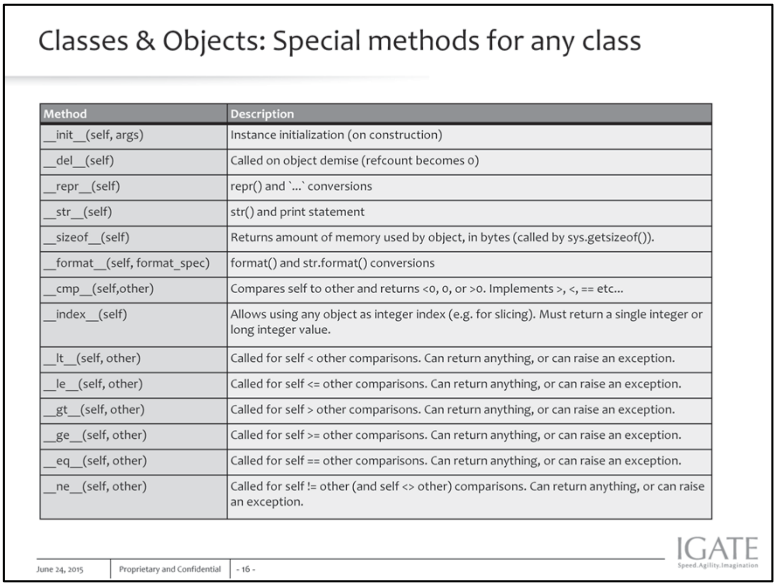


Classes & Objects: Operator Overloading

Operator	Special method	Operator	Special method
self + other	__add__(self, other)	+self	__pos__(self)
self - other	__sub__(self, other)	abs(self)	__abs__(self)
self * other	__mul__(self, other)	~self	__invert__(self) (bitwise)
self / other	__div__(self, other) or __truediv__(self,other) if __future__.division is active.	self += other	__iadd__(self, other)
self // other	__floordiv__(self, other)	self -= other	__isub__(self, other)
self % other	__mod__(self, other)	self *= other	__imul__(self, other)
divmod(self,other)	__divmod__(self, other)	self /= other	__idiv__(self, other) or __itruediv__(self ,other) if __future__.division is in effect.
self ** other	__pow__(self, other)	self //= other	__ifloordiv__(self, other)
self & other	__and__(self, other)	self %= other	__imod__(self, other)
self ^ other	__xor__(self, other)	self **= other	__ipow__(self, other)
self | other	__or__(self, other)	self &= other	__iand__(self, other)
self << other	__lshift__(self, other)	self ^= other	__ixor__(self, other)
self >> other	__rshift__(self, other)	self |= other	__ior__(self, other)
bool(self)	__nonzero__(self) (used in boolean testing)	self <<= other	__ilshift__(self, other)
-self	__neg__(self)	self >>= other	__irshift__(self, other)

Right-hand-side equivalents for all **binary** operators exist (__radd__, __rsub__, __rmul__, __rdiv__, ...).
They are called when class instance is on r-h-s of operator:
-- a + 3 calls __add__(a, 3)   -- 3 + a calls __radd__(a, 3)

June 24, 2015 | Proprietary and Confidential | - 15 -

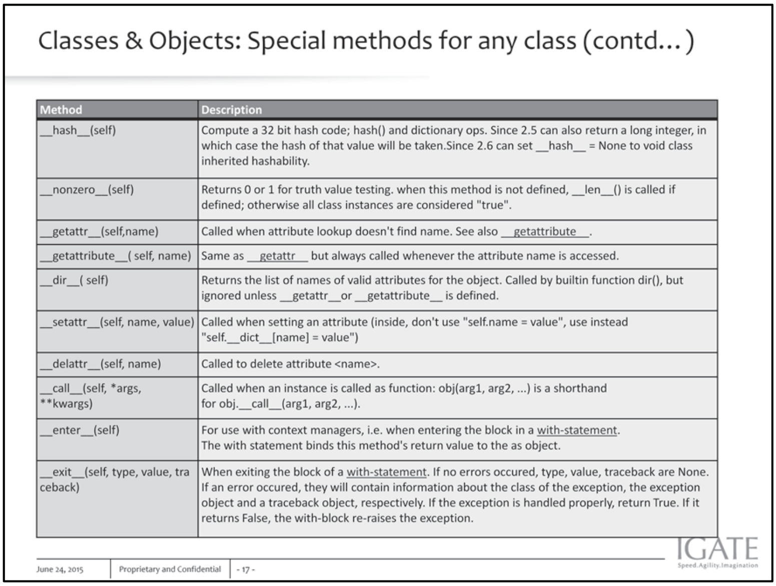


Classes & Objects: Special methods for any class

Method	Description
__init__(self, args)	Instance initialization (on construction)
__del__(self)	Called on object demise (refcount becomes 0)
__repr__(self)	repr() and `...` conversions
__str__(self)	str() and print statement
__sizeof__(self)	Returns amount of memory used by object, in bytes (called by sys.getsizeof()).
__format__(self, format_spec)	format() and str.format() conversions
__cmp__(self,other)	Compares self to other and returns <0, 0, or >0. Implements >, <, == etc...
__index__(self)	Allows using any object as integer index (e.g. for slicing). Must return a single integer or long integer value.
__lt__(self, other)	Called for self < other comparisons. Can return anything, or can raise an exception.
__le__(self, other)	Called for self <= other comparisons. Can return anything, or can raise an exception.
__gt__(self, other)	Called for self > other comparisons. Can return anything, or can raise an exception.
__ge__(self, other)	Called for self >= other comparisons. Can return anything, or can raise an exception.
__eq__(self, other)	Called for self == other comparisons. Can return anything, or can raise an exception.
__ne__(self, other)	Called for self != other (and self <> other) comparisons. Can return anything, or can raise an exception.

June 24, 2015 | Proprietary and Confidential | - 16 -

IGATE
Speed.Agility.Imagination

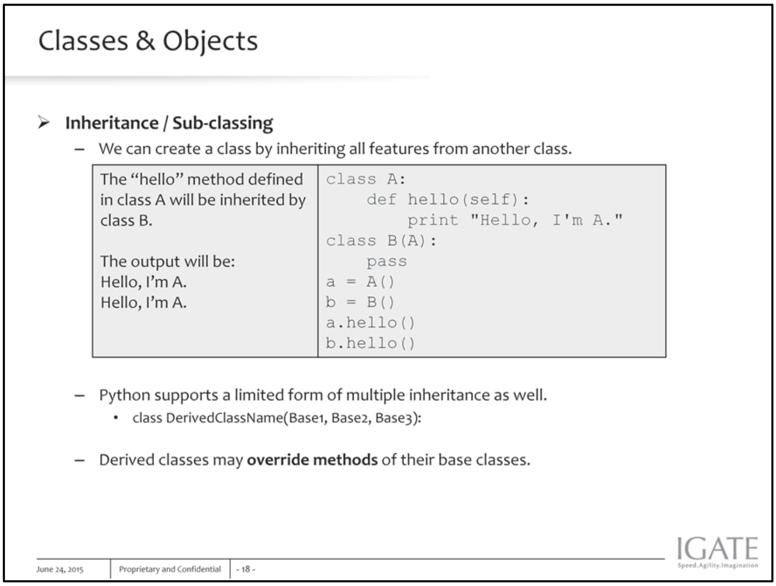


Classes & Objects: Special methods for any class (contd…)

Method	Description
__hash__(self)	Compute a 32 bit hash code; hash() and dictionary ops. Since 2.5 can also return a long integer, in which case the hash of that value will be taken.Since 2.6 can set __hash__ = None to void class inherited hashability.
__nonzero__(self)	Returns 0 or 1 for truth value testing. when this method is not defined, __len__() is called if defined; otherwise all class instances are considered "true".
__getattr__(self,name)	Called when attribute lookup doesn't find name. See also __getattribute__.
__getattribute__(self, name)	Same as __getattr__ but always called whenever the attribute name is accessed.
__dir__(self)	Returns the list of names of valid attributes for the object. Called by builtin function dir(), but ignored unless __getattr__or __getattribute__ is defined.
__setattr__(self, name, value)	Called when setting an attribute (inside, don't use "self.name = value", use instead "self.__dict__[name] = value")
__delattr__(self, name)	Called to delete attribute <name>.
__call__(self, *args, **kwargs)	Called when an instance is called as function: obj(arg1, arg2, ...) is a shorthand for obj.__call__(arg1, arg2, ...).
__enter__(self)	For use with context managers, i.e. when entering the block in a with-statement. The with statement binds this method's return value to the as object.
__exit__(self, type, value, traceback)	When exiting the block of a with-statement. If no errors occured, type, value, traceback are None. If an error occured, they will contain information about the class of the exception, the exception object and a traceback object, respectively. If the exception is handled properly, return True. If it returns False, the with-block re-raises the exception.

June 24, 2015 | Proprietary and Confidential | - 17 -

IGATE
Speed.Agility.Imagination



Classes & Objects

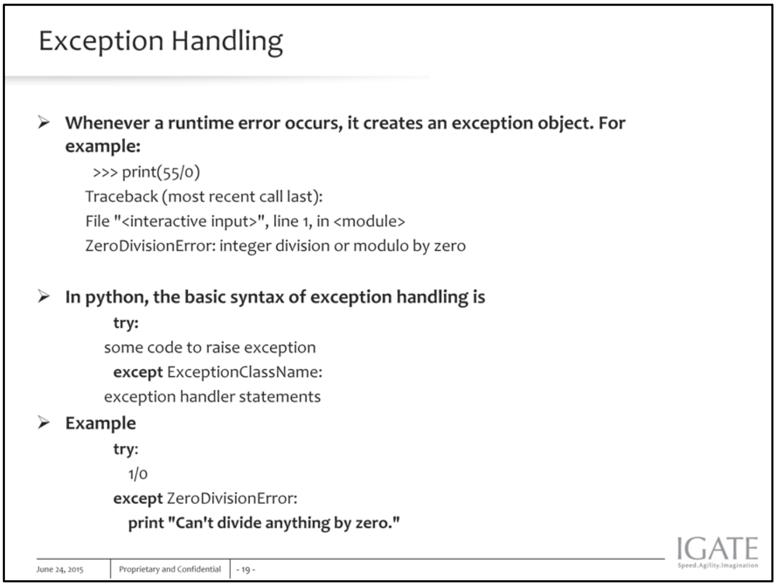
- **Inheritance / Sub-classing**
  - We can create a class by inheriting all features from another class.

The “hello” method defined in class A will be inherited by class B.

The output will be:
Hello, I’m A.
Hello, I’m A.

```
class A:
    def hello(self):
        print "Hello, I'm A."
class B(A):
    pass
a = A()
b = B()
a.hello()
b.hello()
```

  - Python supports a limited form of multiple inheritance as well.
    - class DerivedClassName(Base1, Base2, Base3):
  - Derived classes may **override methods** of their base classes.



Exception Handling

- **Whenever a runtime error occurs, it creates an exception object. For example:**

```
 >>> print(55/0)
Traceback (most recent call last):
File "<interactive input>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

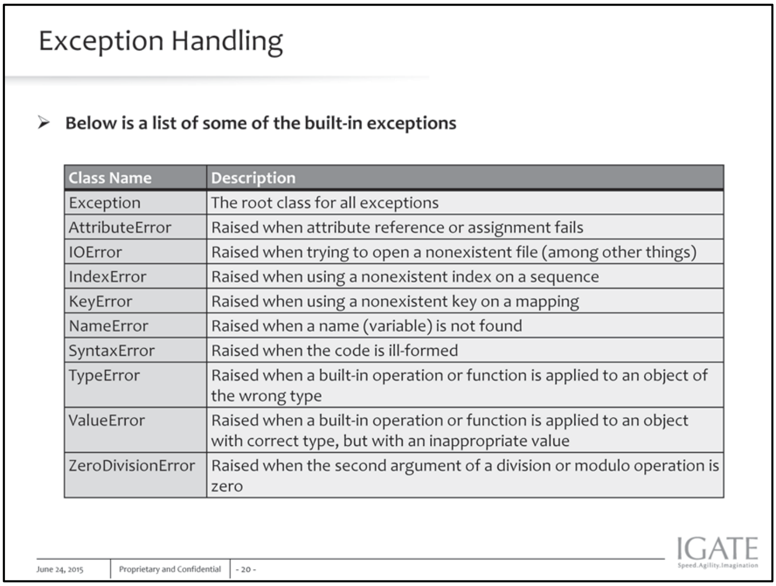
- **In python, the basic syntax of exception handling is**

```
  try:
some code to raise exception
  except ExceptionClassName:
exception handler statements
```

- **Example**

```
try:
   1/0
except ZeroDivisionError:
   print "Can't divide anything by zero."
```

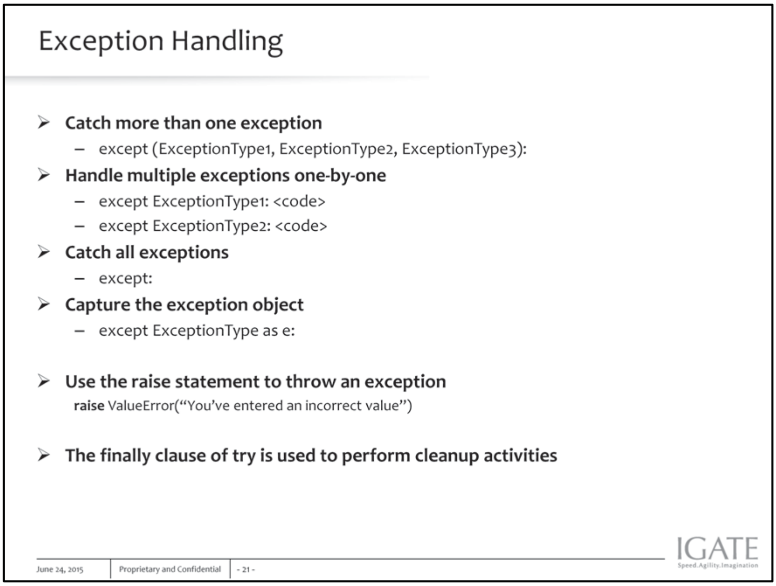
June 24, 2015 | Proprietary and Confidential | - 19 -



Exception Handling

- **Below is a list of some of the built-in exceptions**

Class Name	Description
Exception	The root class for all exceptions
AttributeError	Raised when attribute reference or assignment fails
IOError	Raised when trying to open a nonexistent file (among other things)
IndexError	Raised when using a nonexistent index on a sequence
KeyError	Raised when using a nonexistent key on a mapping
NameError	Raised when a name (variable) is not found
SyntaxError	Raised when the code is ill-formed
TypeError	Raised when a built-in operation or function is applied to an object of the wrong type
ValueError	Raised when a built-in operation or function is applied to an object with correct type, but with an inappropriate value
ZeroDivisionError	Raised when the second argument of a division or modulo operation is zero



Exception Handling

- **Catch more than one exception**
  - except (ExceptionType1, ExceptionType2, ExceptionType3):
- **Handle multiple exceptions one-by-one**
  - except ExceptionType1: <code>
  - except ExceptionType2: <code>
- **Catch all exceptions**
  - except:
- **Capture the exception object**
  - except ExceptionType as e:

- **Use the raise statement to throw an exception**

  raise ValueError(“You’ve entered an incorrect value”)

- **The finally clause of try is used to perform cleanup activities**

June 24, 2015 | Proprietary and Confidential | - 21 -

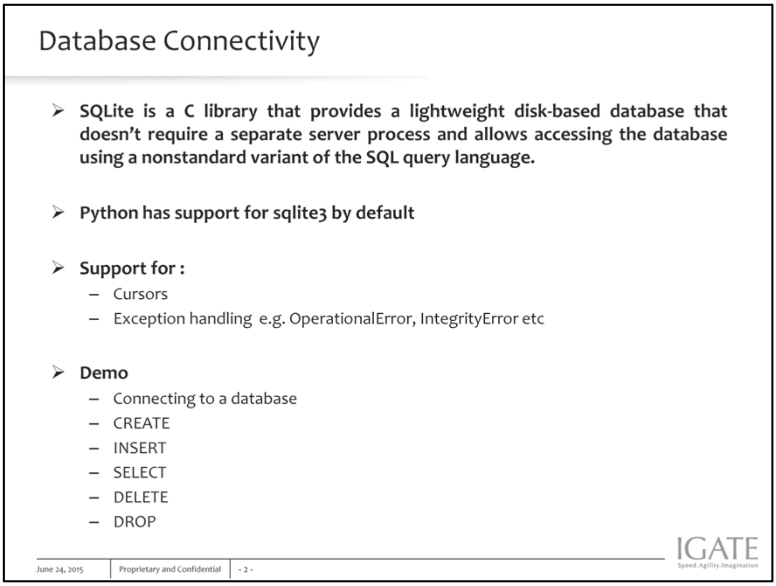
IGATE Speed.Agility.Imagination

Python Training

A basic overview

IGATE

Speed.Agility.Imagination

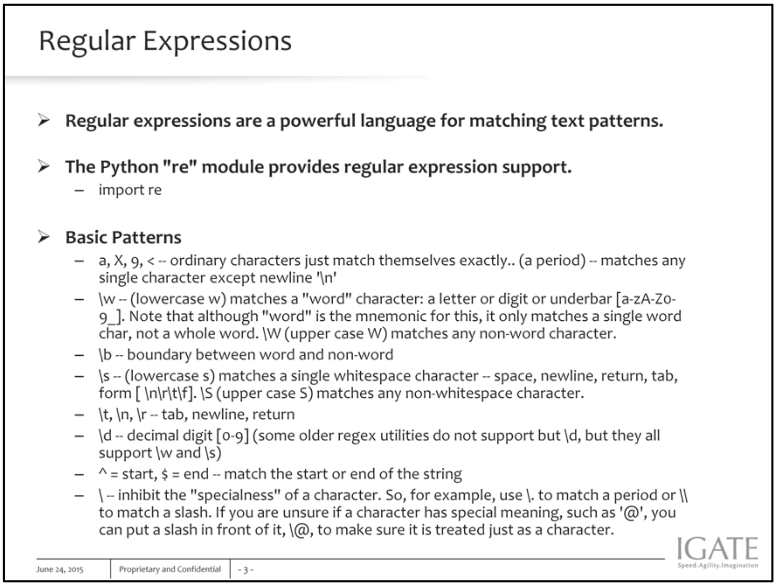


Database Connectivity

- SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.
- Python has support for sqlite3 by default
- Support for :
  - Cursors
  - Exception handling e.g. OperationalError, IntegrityError etc
- Demo
  - Connecting to a database
  - CREATE
  - INSERT
  - SELECT
  - DELETE
  - DROP

June 24, 2015 | Proprietary and Confidential | - 2 -

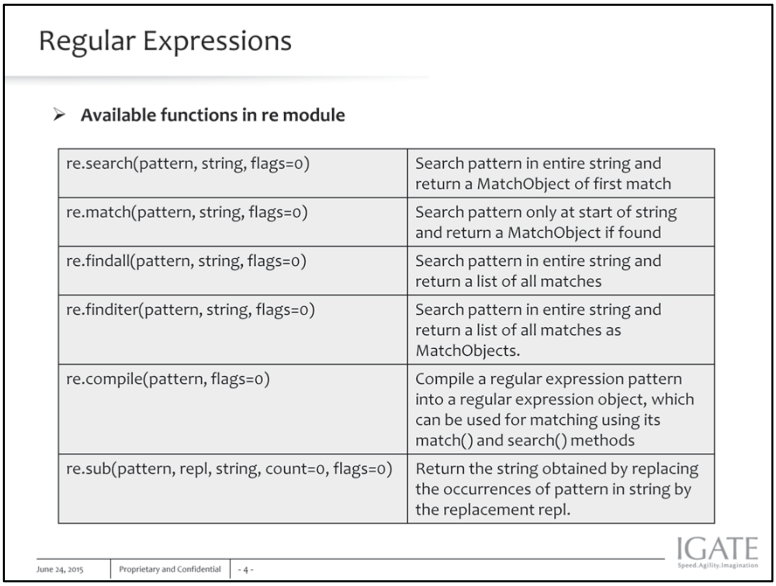
Demo of SQLite data base connectivity, CREATE, INSERT, SELECT, DELETE, DROP to be given



Regular Expressions

- Regular expressions are a powerful language for matching text patterns.
- The Python "re" module provides regular expression support.
  - import re
- Basic Patterns
  - a, X, 9, < -- ordinary characters just match themselves exactly.. (a period) -- matches any single character except newline '\n'
  - \w -- (lowercase w) matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_]. Note that although "word" is the mnemonic for this, it only matches a single word char, not a whole word. \W (upper case W) matches any non-word character.
  - \b -- boundary between word and non-word
  - \s -- (lowercase s) matches a single whitespace character -- space, newline, return, tab, form [\n\r\t\f]. \S (upper case S) matches any non-whitespace character.
  - \t, \n, \r -- tab, newline, return
  - \d -- decimal digit [0-9] (some older regex utilities do not support but \d, but they all support \w and \s)
  - ^ = start, $ = end -- match the start or end of the string
  - \ -- inhibit the "specialness" of a character. So, for example, use \. to match a period or \\ to match a slash. If you are unsure if a character has special meaning, such as '@', you can put a slash in front of it, \@, to make sure it is treated just as a character.

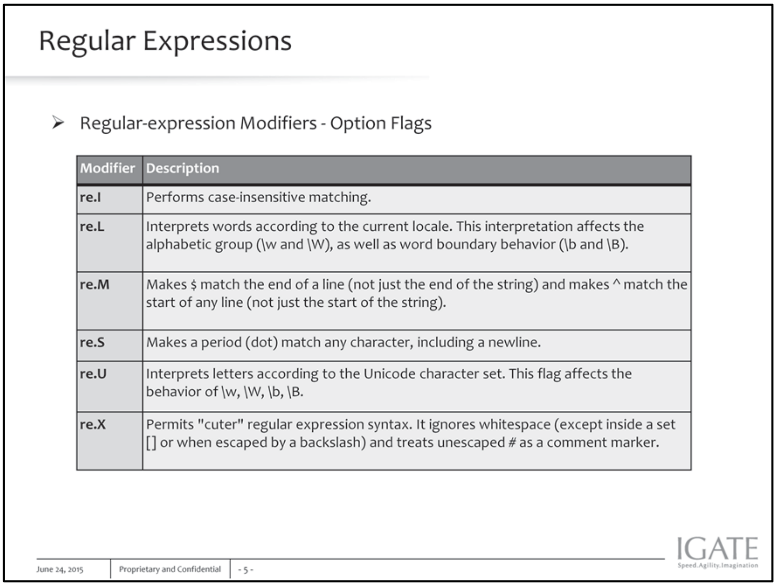
June 24, 2015 | Proprietary and Confidential | - 3 -



Regular Expressions

- **Available functions in re module**

re.search(pattern, string, flags=0)	Search pattern in entire string and return a MatchObject of first match
re.match(pattern, string, flags=0)	Search pattern only at start of string and return a MatchObject if found
re.findall(pattern, string, flags=0)	Search pattern in entire string and return a list of all matches
re.finditer(pattern, string, flags=0)	Search pattern in entire string and return a list of all matches as MatchObjects.
re.compile(pattern, flags=0)	Compile a regular expression pattern into a regular expression object, which can be used for matching using its match() and search() methods
re.sub(pattern, repl, string, count=0, flags=0)	Return the string obtained by replacing the occurrences of pattern in string by the replacement repl.



Regular Expressions

- Regular-expression Modifiers - Option Flags

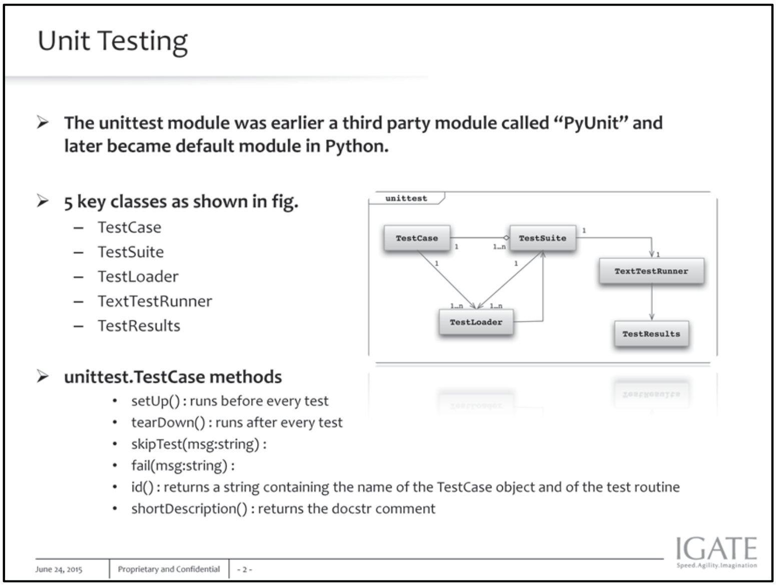
Modifier	Description
re.I	Performs case-insensitive matching.
re.L	Interprets words according to the current locale. This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior (\b and \B).
re.M	Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string).
re.S	Makes a period (dot) match any character, including a newline.
re.U	Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
re.X	Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash) and treats unescaped # as a comment marker.

Python Training

A basic overview

IGATE

Speed.Agility.Imagination



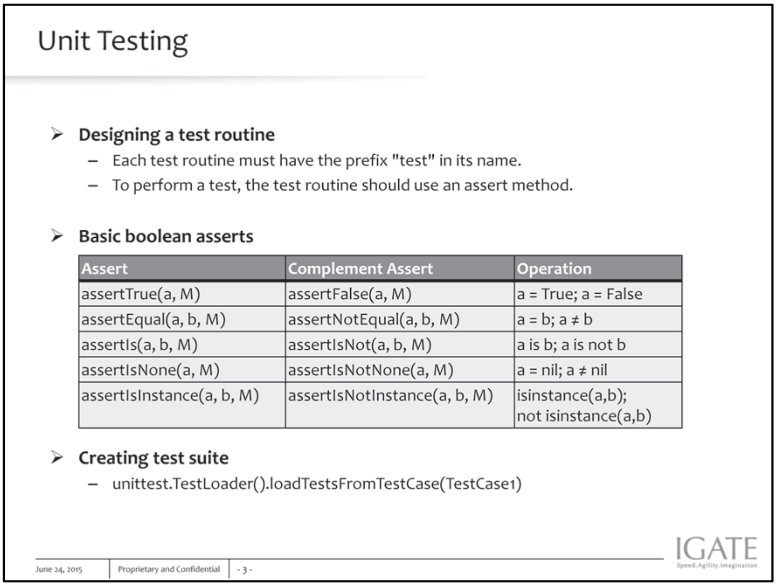
Unit Testing

- The unittest module was earlier a third party module called “PyUnit” and later became default module in Python.
- 5 key classes as shown in fig.
  - TestCase
  - TestSuite
  - TestLoader
  - TextTestRunner
  - TestResults

- **unittest.TestCase methods**
  - setUp() : runs before every test
  - tearDown() : runs after every test
  - skipTest(msg:string) :
  - fail(msg:string) :
  - id() : returns a string containing the name of the TestCase object and of the test routine
  - shortDescription() : returns the docstr comment

June 24, 2015 | Proprietary and Confidential | - 2 -

IGATE Speed.Agility.Imagination



Unit Testing

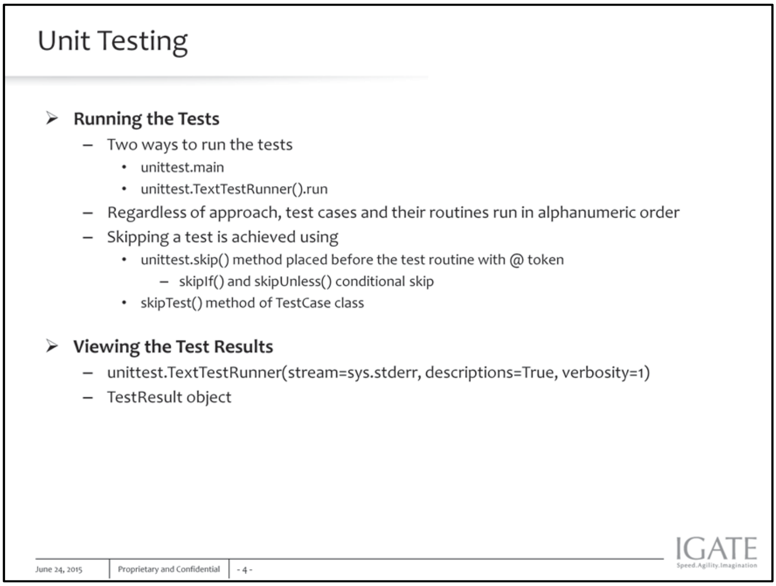
- **Designing a test routine**
  - Each test routine must have the prefix "test" in its name.
  - To perform a test, the test routine should use an assert method.

- **Basic boolean asserts**

Assert	Complement Assert	Operation
assertTrue(a, M)	assertFalse(a, M)	a = True; a = False
assertEqual(a, b, M)	assertNotEqual(a, b, M)	a = b; a ≠ b
assertIs(a, b, M)	assertIsNot(a, b, M)	a is b; a is not b
assertIsNone(a, M)	assertIsNotNone(a, M)	a = nil; a ≠ nil
assertIsInstance(a, b, M)	assertIsNotInstance(a, b, M)	isinstance(a,b); not isinstance(a,b)

- **Creating test suite**
  - unittest.TestLoader().loadTestsFromTestCase(TestCase1)

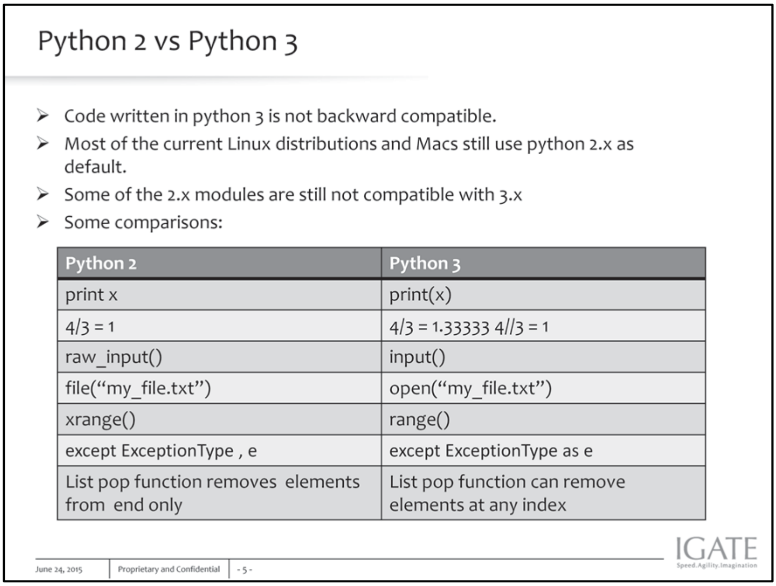
June 24, 2015 | Proprietary and Confidential | - 3 -



Unit Testing

- **Running the Tests**
  - Two ways to run the tests
    - unittest.main
    - unittest.TextTestRunner().run
  - Regardless of approach, test cases and their routines run in alphanumeric order
  - Skipping a test is achieved using
    - unittest.skip() method placed before the test routine with @ token
      - skipIf() and skipUnless() conditional skip
    - skipTest() method of TestCase class
- **Viewing the Test Results**
  - unittest.TextTestRunner(stream=sys.stderr, descriptions=True, verbosity=1)
  - TestResult object

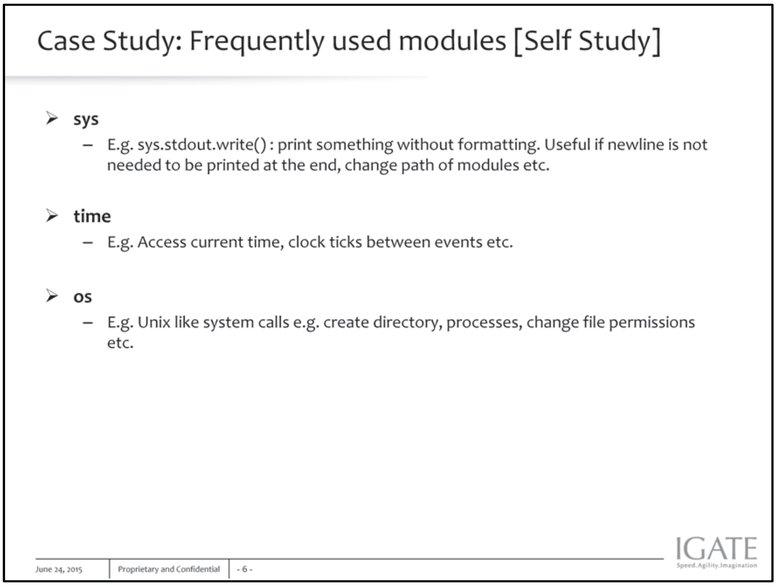
June 24, 2015 | Proprietary and Confidential | - 4 -



Python 2 vs Python 3

- Code written in python 3 is not backward compatible.
- Most of the current Linux distributions and Macs still use python 2.x as default.
- Some of the 2.x modules are still not compatible with 3.x
- Some comparisons:

Python 2	Python 3
print x	print(x)
4/3 = 1	4/3 = 1.33333 4//3 = 1
raw_input()	input()
file(“my_file.txt”)	open(“my_file.txt”)
xrange()	range()
except ExceptionType , e	except ExceptionType as e
List pop function removes elements from end only	List pop function can remove elements at any index



Case Study: Frequently used modules [Self Study]

- **sys**
  - E.g. sys.stdout.write() : print something without formatting. Useful if newline is not needed to be printed at the end, change path of modules etc.
- **time**
  - E.g. Access current time, clock ticks between events etc.
- **os**
  - E.g. Unix like system calls e.g. create directory, processes, change file permissions etc.

June 24, 2015 | Proprietary and Confidential | - 6 -

IGATE Speed.Agility.Imagination