Are you ready for the future?

python™

High Performance Solutions, Backend Development and Integration Services, Embedded Software Development, Big Data (Visualization, Architecture, Science), Business Intelligence & Analytics

Globant
we are ready

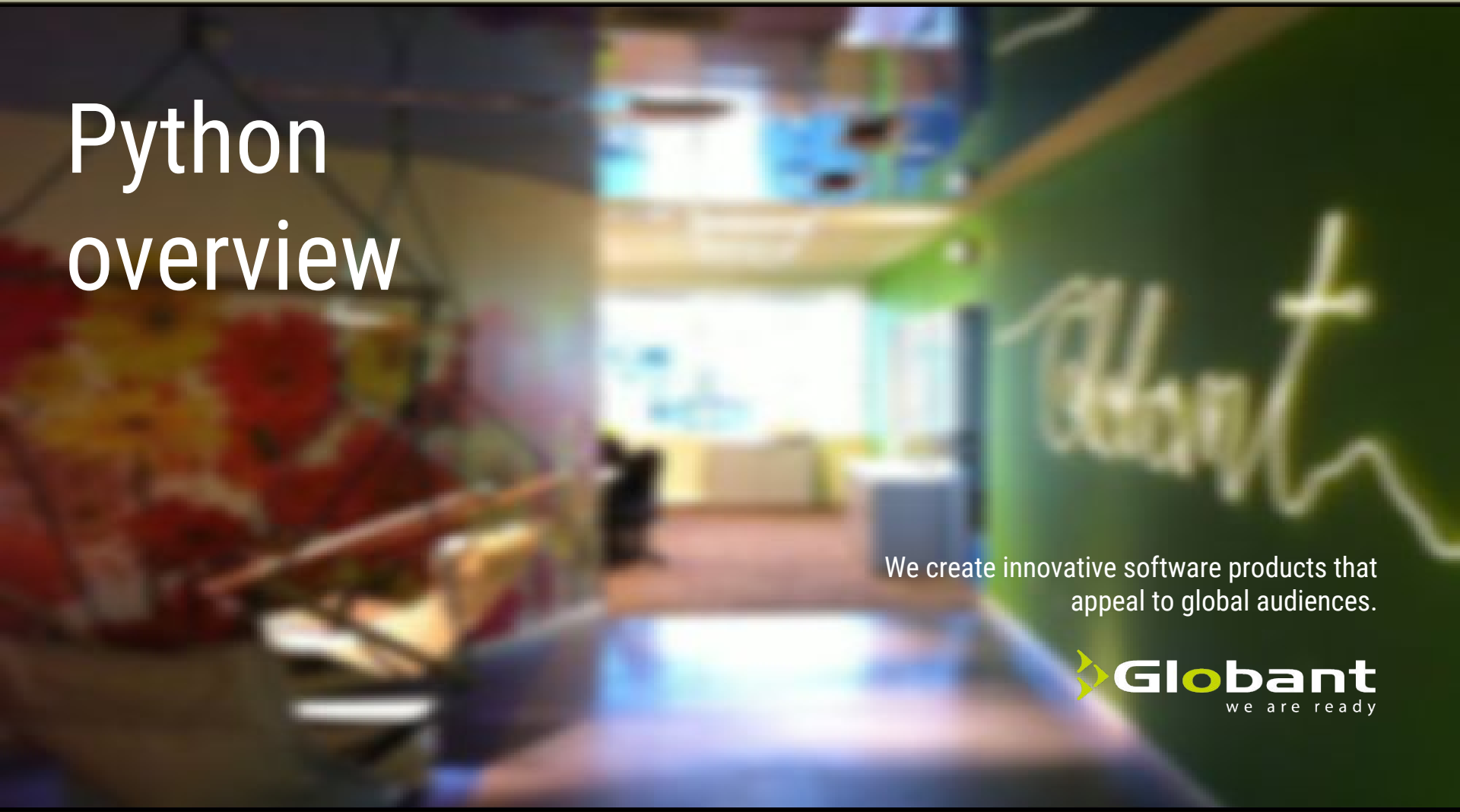
BIG DATA &
HIGH PERFORMANCE

DEC 2014

Objectives

- Python overview
- Python setup and tools
- Python Language basics*
- References

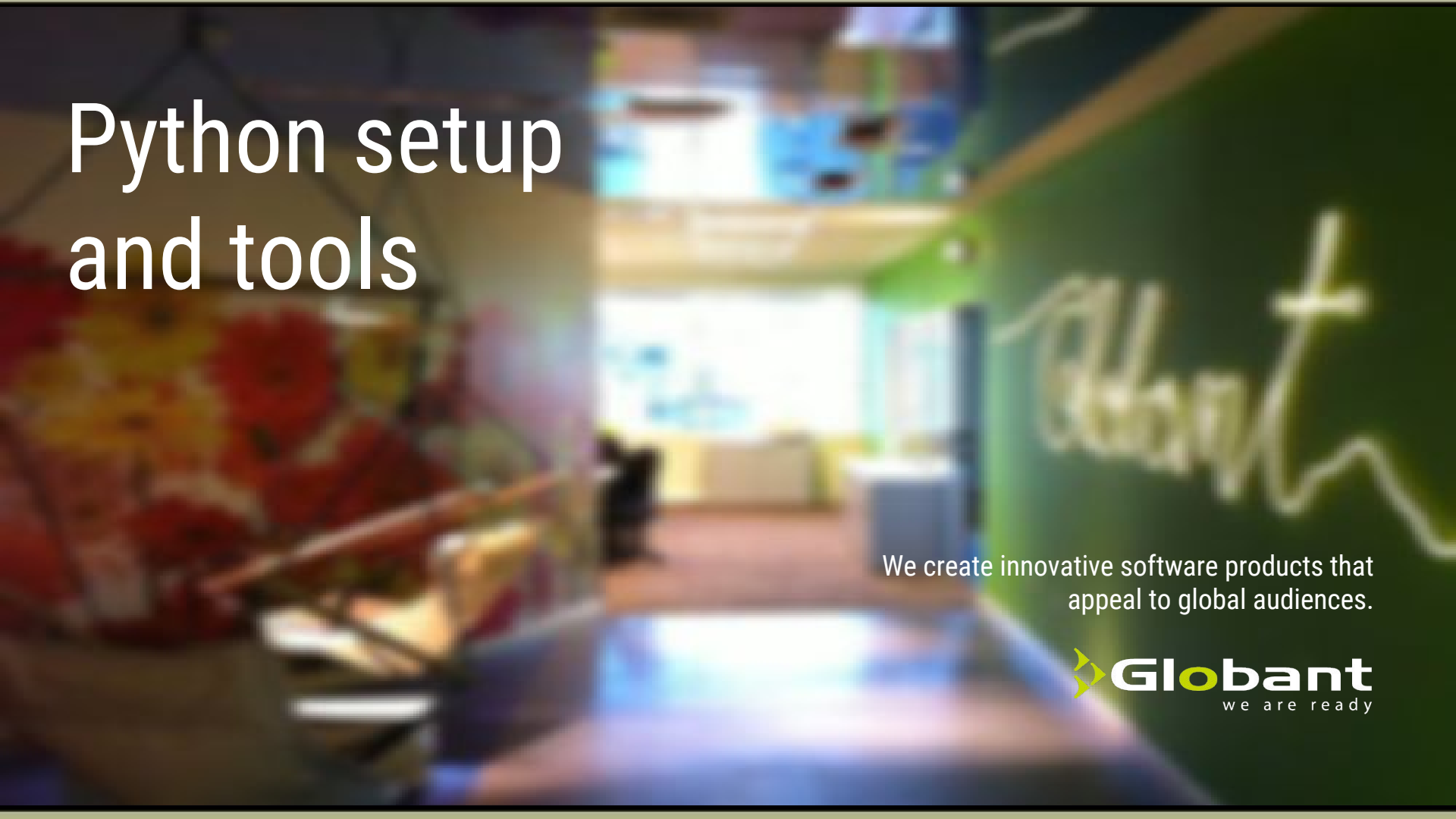
Python overview



Overview

- Useful to coordinate processes and build execution pipelines
- Good replacement for *NIX shell scripting !
- Straightforward syntax, emphasizes readability
- Portable
- Object-Oriented
- Dynamically typed
- Powerful for text manipulation
- Broad range of modules (libraries) for all purposes.
- Two main versions 2 and 3 (very distinct, we'll use 2)

Python setup and tools



Setup and tools

- Language binaries:

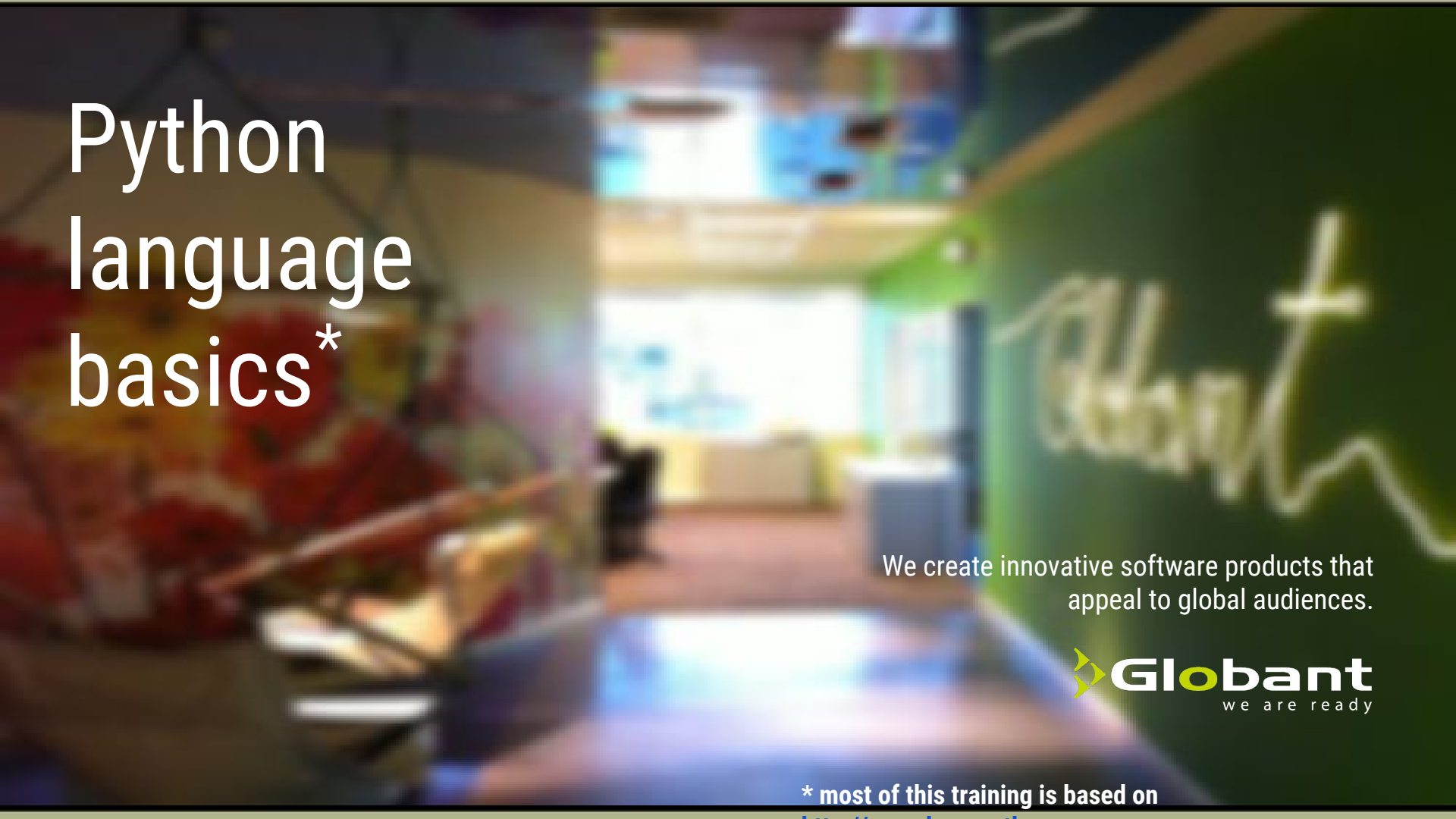
https://www.python.org/downloads/

make sure to use version 2.7

- Spyder : A good IDE !!

http://code.google.com/p/spyderlib/

Python language basics*



* most of this training is based on

Hello World!

Writing to console

```
# hash symbol comments code
print "Hello World!"
```

Indentation (spaces or tabs but NEVER MIX, keep same size)

```
x = 1
# indentation defines blocks, no braces needed
if x == 1:
    # here indented at four spaces.
    print "x is 1."
```

To execute it, either:

- From shell, run `python` and type code there.
- Save code to helloworld.py and run in shell `python helloworld.py`

Variables, basic types and casting

Strings

```
mystring = 'hello'
mystring = "hello"
mystring = str(myint)
```

Integer

```
myint = 7
myint = int("7")
```

Float

```
myfloat = 7.0
myfloat = float(7)
```

Boolean

```
mybool = True
mybool = False
```

Multiple assignation

```
var1, var2, var3 = 3, 100.1, 'hi'
```

Dynamic types

```
var1 = 'hello'      #creates a string
var1 = 5            #changes to int
var1 = 5.0          #changes to float
```

The NoneType

```
# None indicates absence of a value,
# similar to NULL or null in other
# languages
no_value = None
```

Lists

Dynamic-size arrays. Contain any type of object

Indexes start at ZERO

List Example

```
mylist = []
mylist.append(1)
mylist.append("dos")
mylist.insert(2, "tres")
mylist.append(4)
print mylist[0]   # prints 1
print mylist[1]   # prints dos
print mylist[2]   # prints tres
print mylist[3]   # prints 4
```

Iterations

```
# It prints out 1,dos,tres,4
for x in mylist:
    print x
```

Slices (sublists)

```
# It prints out: ['dos','tres']
print mylist[1:3]
```

Operators

Arithmetic:

```
float_number = 1 + 2 * 3 / 4.5

remainder = 11 % 3

squared = 7 ** 2
cubed = 2 ** 3
```

Concatenation:

```
concatenated = "hello" + " " + "world"

# prints hellohello
repeated = "hello" * 2
print repeated
```

Lists:

```
odd_numbers = [1,3,5,7]
even_numbers = [2,4,6,8]
all_numbers = odd_numbers + even_numbers
# now all_numbers is  [1,3,5,7,2,4,6,8]

repeated = [1,2,3] * 2
# now repeated is [1,2,3,1,2,3]
```

String formatting

% operator:

```
# prints out "Hello, John!"
name = "John"
print "Hello, %s!" % name

# prints out "John is 23 years old."
name = "John"
age = 23
print "%s is %d years old." % (name, age)

# prints out "A list: [1, 2, 3]"
mylist = [1,2,3]
print "A list: %s" % mylist
```

Arguments of % operator

`%s` : String (or object with a string representation)

`%d` : Integers

`%f` : Floating point numbers

`%.<number of digits>f` : Floating point with fixed digits

String operations


```
a_string = "Hello world!"

# prints the length of a_string: 12
print len(a_string)

# prints the index of first "o": 4
print a_string.index("o")

# prints the substring between
# indices 3 and 6: "lo w"
print a_string[3:7]

# prints a_string in uppercase
print a_string.upper()

# prints "Hell- w-rld"
print a_string.replace("o", "-")
```

```
# prints True
print a_string.startswith("Hello")

# prints False
print a_string.endswith("asdfasdfasdf")

# breaks a_string into list
# of strings (here space used as separator)
tokens = a_string.split(" ")

# concatenates a list of strings
# (here space used as separator)
a_string = " ".join(tokens)
```

Conditions

```
x = 2
```

Comparison operators:

```
print x == 2      # prints out True
print x != 2      # prints out False
print x >= 3      # prints out False
print x < 3       # prints out True
```

Boolean operators:

```
name = "John"
age = 18
if name == "John" and age >= 17:
   print "Hi John, the army awaits you!"
else:
   print "Your are either not John or under 17."
```

or, **not** are also a boolean operators

“in” operator:

```
if name in ["John", "Rick"]:
    print "Your name is either John or Rick."
else:
    print "I don’t know you"
```

“is” vs. “==” operator:

```
x = [1,2,3]
y = [1,2,3]
print x == y      # Prints True
print x is y      # Prints False
```

Loops

For

```
# prints 2,3,5,7
primes = [2, 3, 5, 7]
for count in primes:
    print count
```

"range" function:

```
# creates a list [0,1,2,3]
sequence = range(4)

# prints 0,1,2,3
for count in range(4):
    print count
```

While

```
# also prints 0,1,2,3
count = 0
while True:
    print count
    count += 1
    if count >= 4:
        break
```

Globant
we are ready

Functions and Classes

FUNCTIONS

Declaration

```
def my_function():
    print "Hi From My Function!"
```

Invocation

```
# prints "Hi From My Function!"
my_function()
```

Arguments and return **value**(**s**)

```
# returns two values:
# a sum and a division of two numbers
def my_sum_and_divide(arg1, arg2):
    return arg1+arg2, arg1/arg2

# prints 12, 5.0
mysum, mydiv = my_sum_and_divide(10,2)
print mysum, mydiv
```

CLASSES

Declaration

```
# ALL members are always public. So be careful
class MyClass:
    variable = "blah"

    # keyword "self" makes the function a metho
    def hello(self, name):  # self
        print "Hi from the class " + name
```

Instantiation

```
myobject = MyClass()

# prints: blah
print myobject.variable

# prints: Hi from the class John
myobject.hello("John")
```

Globant
we are ready

Dictionaries

Key-Value containers

Dictionary Example

```
my_phone_dict = {}
my_phone_dict["John"] = "938477566"
my_phone_dict["Jack"] = "938377264"
my_phone_dict["Jill"] = "947662781"

# Or it also be declared as:
my_phone_dict = {
    "John" : "938477566",
    "Jack" : "938377264",
    "Jill" : "947662781"
}
```

Iterations

```
# It prints the phone numbers
for name, number in my_phone_dict.iteritems():
    print "number of %s is %d" % (name, number)
```

Removing a Key-Value

```
del my_phone_dict["John"]
# or also
my_phone_dict.pop("John")
```

Modules

Files containing python functions with .py extension. They are imported from another module.

```
# importing another full module
import mymodule
# calling one of it's functions
mymodule.function_1()
```

Alternatively (recommended):

```
# importing just a function of a module
from module import function_1
# calling the function
function_1()
```

Useful Standard Modules

```
sys       #access to stdin, stdout, stderr
os        #access to shell and file system
urllib    #http requests (webpages, etc.)
re        #regular expressions
```

Third party Modules can be installed with the tools: easy_install or pip (prefered)

Program arguments and Standard Input

Command line arguments

```
#filename: hello.py
import sys

# prints out True
print "Hello " + sys.argv[1]
```

Run it:

```
$ python hello.py John
```

Output is:

```
Hello John
```

Standard input

```
#filename: hello.py
import sys

# prints out True
for line in sys.stdin:
    print line.strip("\n")
```

Run it with unix pipes:

```
$ echo -e "Hello\nJohn" | python hello.py
```

Output is:

```
Hello
John
```

Debugging Tips

- use of type() function

```
# In a long code, you don't know what type of object feat is
...
...
def a_weird_function(feat):
    lb = LabelBinarizer()
    binarized_data = np.transpose(lb.fit_transform(feat))
    # it's useful to know the datatype of a variable.
    # just add the line
    print type(feat)   # here it prints : <type 'numpy.array'>
    # Go and google up: numpy.array to learn its API.
...
...
```

- if weird syntax errors, perhaps due to a mix of spaces and tabs.

- Before attempting something complicated, google for the pythonic way to do it. I can save you lots of time. The language is very rich.

References

- http://www.codecademy.com/en/tracks/python
  Interactive Tutorial and really comprehensive. Try to follow it as much as possible.

- http://www.learnpython.org
  Interactive. The “basic” part covers this tutoria. Look for the “advanced part”

- http://repl.it/languages/Python
  Interactive python editor