Digital Mass #4: Python

Impariamo a programmare con

A cura di **Federico Capoano**

Chi siamo?

Federico Capoano

Web Geek

Membro attivo di Ninux

@nemesisdesign su twitter

http://nemesisdesign.net/

Claudio Pisa

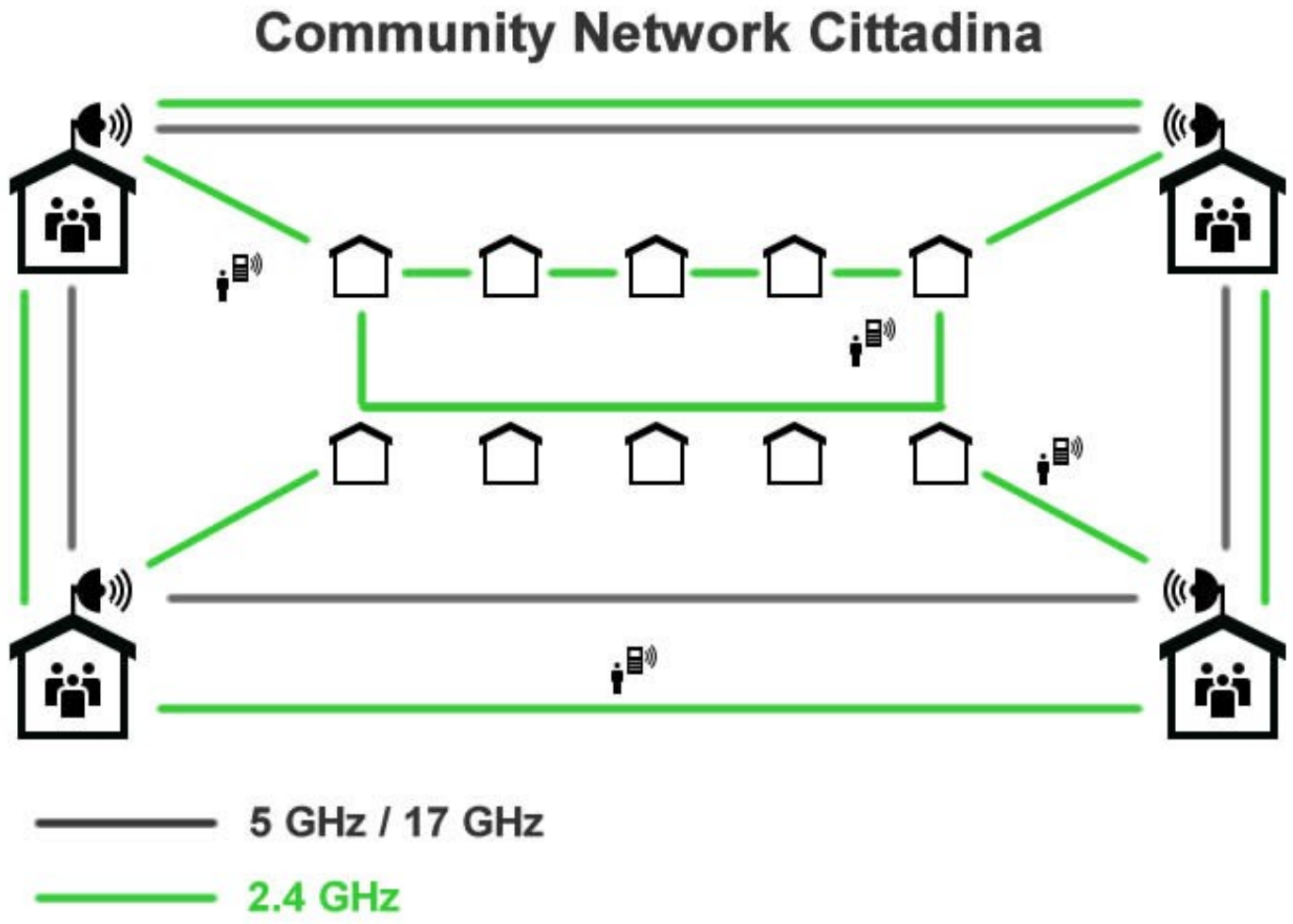
Linux & Networking Geek

Uno dei fondatori di Ninux

@cl4u2 su twitter

La Community **Ninux.org** e l'associazione **Fusolab ONLUS**

Due parole su Ninux



Due parole su Ninux

La mission di ninux

1. Costruire una rete decentralizzata di proprietà dei partecipanti

2. Aiutare le realtà colpite da Digital Divide condividendo gratuitamente la connessione ad internet della community

3. Diffondere gli ideali della libertà di comunicazione, la neutralità della rete, la filosofia del software libero, la collaborazione volontaria per un fine comune e la condivisione dei saperi

Obiettivi di questo workshop

Vorrei che chi uscisse da questa stanza...

- **Panoramica generale di python 2.X**
- **Approfondire in autonomia**
- **Stimolare la vostra curiosità!**
- **Prossimi appuntamenti: Scapy e Django**

Cos'è Python?

- Linguaggio di programmazione a oggetti di alto livello
- Semplice da imparare, semplice da leggere
- Multiuso: scripting, desktop, web, networking, scientifico
- Multipiattaforma (Unix, Windows)
- Multiparadigma: object-oriented, funzionale, riflessiva
- Pseudocompilato / precompilato
- Open source

The Zen of Python

La filosofia di Python riassunta in una poesia

Se avete unix, aprite la command line e digitate

```
$> python

>>> import this
```

Se non avete un sistema unix... **installatelo! LOL**

Oppure cercate su google: **“The Zen of Python”**

“Pythonic”

Cosa significa “Pythonic” ?

Imparare Python è come imparare una lingua:

non biosgna tradurre in modo letterale!

Pythonic è un modo di programmare tipico di python.

Unpythonic è quando si tenta di usare uno stile di programmazione che è simile ad altri linguaggi C-like.

“Pythonic”

- Alcuni idiomi sono tipici in python
- Altri non lo sono affatto e vanno evitati
- Alcuni costrutti sono stati volutamente esclusi dal linguaggio
  Esempi: switch e operatore ternario
- Altre cose vengono gestite automaticamente
  Esempio: garbage collection;

Quando si apprende a scrivere codice “Pythonic” non si sente più la mancanza di ciò che è stato escluso o che viene gestito automaticamente.

Installare Python

Sui sistemi Unix (Linux e Mac) Python 2.x è pre installato.

Sui sistemi Windows va installato. Si può scaricare un installer binario da **www.python.org** .

Potete verificare se python è installato digitando il comando:

```
$> python
```

Virtual Environment

- Se non volete “sporcare” l'installazione di python del vostro sistema

- Se avete bisogno di sviluppare programmi con diverse versioni di Python

- Se volete lavorare nel modo giusto!

Usate **Python Virtual Environment**!

Virtual Environment

Cos'è un “Virtual Environment”?

E' un installazione di Python isolata dal resto del sistema operativo

Su Linux (Ubuntu)

```
$ sudo apt-get install python-virtualenv
$ cd /home/myuser/myproject
$ virtualenv NOME_ENV
$ source NOME_ENV/bin/activate
$ # ready to go!
$ python
```

La Shell Interattiva

E' possibile utilizzare python dalla riga di comando.

Epperchè mai???

- Smanettare ed imparare
- Testare il codice prima di scrivere minchiate ;-)
- Debugging rapido
- Because we can

La Shell Interattiva

E' difficilissimo!

```
$ python
Python 2.7.2 (default, Jun 20 2012, 16:23:33)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-
418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.

>>> print “Hello World!”
Hello World!
```

L'indentazione

In Python i blocchi sono delimitati dall'indentazione;

Non ci sono parentesi graffe o “end” keywords;

Sono ammessi sia spazi che tab, purchè non siano mischiati;

Ovvero in un file o ci sono solo tab o ci sono solo spazi;

La convenzione è 4 spazi, vi consiglio di seguirla!

Se non indentate correttamente l'interprete vi restituirà un errore di tipo **“IndentationError”.**

L'indentazione

Esempio:

```
>>> if True:
...     print “My First Block!”
...
My First Block!
```

L'indentazione

Esempio:

```
>>>       print “indentazione sbagliata”
  File "<stdin>", line 1
    print 'indentazione sbagliata'
    ^
IndentationError: unexpected indent
```

Ed il punto e virgola?

Il punto e virgola è necessario solo quando si scrivono più “statements” (dichiarazioni) su una sola riga:

```
>>> # one line
>>> x = 'Prova one line'; print x;
Prova one line

>>> x = 'Multiline'
>>> print x
Multiline
```

Booleani

Forse avete appena notato una cosa: la keyword “**True**”.

In python i valori “vero” e “falso” si scrivono
“**True**” e “**False**” con la prima lettera maiuscola

Sono espressioni vere ad esempio:
- 1
- 'stringa'

Mentre queste sono alcune espressioni false:
- 0
- ''

Booleani

Esempio:

```
>>> if 1: print “Vero”;
>>> Vero
>>> if 'stringa': print “Vero”;
>>> Vero

>>> if 0: print 'falso, non stampa nulla';
...
>>> if '': print 'falso, non stampa nulla';
...
```

Ed il punto e virgola?

Il punto e virgola è necessario solo quando si scrivono più “statements” (dichiarazioni) su una sola riga:

```
>>> # one line
>>> x = 'Prova one line'; print x;
Prova one line

>>> x = 'Multiline'
>>> print x
Multiline
```

Commenti nel codice

In python esistono solo commenti a riga singola

```
>>> # questo è un commento
```

Le funzioni

In python le funzioni si dichiarano così:

```
>>> def hello(name):
>>>     print “Hello %s” % name

>>> hello(“Federico”)
Hello Federico
```

Cos'era <“Hello %s” % name> ?

E' una funzionalità per la formattazione delle stringhe comunemente utilizzata in python.

Le funzioni

Parametri opzionali:

```
>>> def hello(first_name, last_name=None):
>>>   if last_name:
>>>     name = “%s %s” % (first_name, last_name)
>>>   else:
>>>     name = first_name
>>>   print “Hello %s” % name
...
>>> hello('Paolo', 'Rossi')
Hello Paolo Rossi
>>> hello('Teo')
Teo
```

Tutto è un oggetto

In Python tutto è un oggetto, anche le funzioni!

```
>>> hello
>>> <function hello at 0x10f12cb18>
>>> hello.__name__
hello

>>> # utilizziamo il metodo “split” delle stringhe
>>> 'stringa1, stringa2, stringa3'.split(', ')
>>> # divide la stringa in acccordo col delimitatore
>>> ['stringa1', 'stringa2', 'stringa3']
```

Docstrings

Una delle cose più fighe di python è che si può scrivere la documentazione nel codice.

```
>>> def hello(first_name, last_name=None):
>>>   """ Stampa nome e cognome oppure solo il nome """
>>>   if last_name:
>>>     name = "%s %s" % (first_name, last_name)
>>>   else:
>>>     name = first_name
>>>   print "Hello %s" % name
...
>>> hello.__doc__
' Stampa nome e cognome oppure solo il nome '
```

Si possono scrivere docstrings solo per funzioni e metodi (cosa sono i metodi lo vedremo più avanti).

Curiosare in giro

Segnatevi queste due funzioni base del linguaggio!

- **dir()** - restituisce una lista dei metodi e degli attributi di un oggetto
- **help()** - apre un editor testuale read-only con la documentazione dell'oggetto

```
>>> dir(hello)
['__call__', '__class__', '__closure__', '__code__',
'__defaults__', '__delattr__', '__dict__',
'__doc__'...

>>> help(hello)
```

Tipi di dato essenziali

I tipi di dato con cui dovete assolutamente divenire familiari in python sono:

- Liste
- Tuple
- Dizionari

Liste

Una lista di valori indicizzati con una chiave numerica progressiva che parte da 0.

- se conoscete PHP: sono Array numerici
- se conoscete Java: tipo gli Array ma con più funzionalità

```
>>> lista = ['banana', 'pera', 'mela']
>>> type(lista)
<type 'list'>
>>> lista[0]
'banana'
>>> # divertitevi a vedere i metodi disponibili
>>> dir(lista)
```

Liste

```
>>> help(lista.append)
>>> lista.append('arancia')
>>> lista
['banana', 'pera', 'mela', 'arancia']
# operatore “in”, una delle cose che più mi piacciono
'arancia' in lista
True

Scorrere una lista è davvero difficile..!
>>> for frutto in lista: # ai cicli ci arriviamo dopo
>>>   print frutto
banana
pera
mela
arancia
```

List Slicing!

```
>>> # le prime 2
>>> lista[0:2]
['banana', 'pera']
>>> lista[:2]
['banana', 'pera']
>>> # dalla 3° in poi
>>> lista[2:]
['mela', 'arancia']
>>> # l'ultima
>>> lista[-1]
'arancia'
>>> # insomma avete capito :D
>>> lista[-3:-1]
['pera', 'mela']
```

Tuple

Sono liste immutabili: ovvero non hanno metodi per l'aggiunta di elementi.

```
>>> tupla = ('banana', 'pera', 'mela')
>>> type(lista)
<type 'tuple'>
>>> tupla[0]
'banana'
>>> tupla[0] = 'prova'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assignment
```

Tuple

Anche le tuple supportano lo slicing:

```
>>> tupla[0:2]
('banana', 'pera')
```

Attenzione perchè una tupla da un elemento solo si scrive così:

```
>>> ('un elemento',)
```

Perchè altrimenti...

```
>>> ('una stringa!')
'una stringa!'
>>> type(('una stringa!'))
<type 'str'>
```

Dizionari

Lista di coppie chiavi – valori.

- se conoscete PHP: sono simili agli array associativi
- se conoscete Perl o Ruby: sono simili agli hash
- se conoscete Java: sono simili alle istanze della classe Hashtable

```
>>> dizionario = {'chiave1': 'valore1', 'chiave2': 'valore2'}
>>> type(dizionario)
<type 'dict'>
>>> dizionario['chiave1']
>>> 'valore1'
>>> dizionario.get('non_esiste', False)
False
>>> dizionario['non_esiste']
KeyError: 'non_esiste'
```

Dizionari

Esploriamo i metodi

```
>>> dizionario = { 'chiave1': 'valore1', 'chiave2': 'valore2' }
>>> dir(dizionario)
>>> dizionario.items()
>>> dizionario.keys()
>>> dizionario.pop('chiave1')
>>> help(dizionario.pop)
```

Come scorrere un dizionario:

```
>>> for key, value in dizionario.items():
>>>     print "%s: %s" % (key, value)
```

Operatori principali

Vediamo gli operatori essenziali:

- and (logico)
- or (logico)
- not (negazione)
- In (presenza)
- Is (uguaglianza)
- > (maggiore)
- < (minore)
- >= (maggiore o uguale)
- <= (minore o uguale)
- == (uguale)
- != (diverso)

Costrutto if

Se una condizione specificata è vera l'interprete esegue il codice nel blocco

```
>>> a = 10
>>> b = 1
>>> if a > b:
>>>     print “a è maggiore di b”
>>> elif a == b:
>>>     print “a è uguale a b”
>>> else:
>>>     print “a è minore di b”
```

Si possono usare varie combinazioni diverse di operatori per creare espressioni complesse

Cicli for

Il ciclo “for” in python serve per scorrere liste, tuple, dizionari e altri oggetti simili chiamati “iterators”.

```
>>> persone = ['marco', 'adriano', 'valeria', 'tina']
>>> for persona in persone:
>>>     print persona
marco
adriano
valeria
tina
```

Si può passare all'iterazione successiva utilizzando la keyword “continue” e si può uscire dal ciclo utilizzando al keyword “break”.

Cicli while

Un ciclo while è un blocco di codice che si ripete fin quando la condizione specificata dopo la keyword while è vera.

```
>>> n = 1
>>> while n <= 10:
...   print n
...   n += 1
```

Si può passare all'iterazione successiva utilizzando la keyword “continue” e si può uscire dal ciclo utilizzando al keyword “break”.

Cosa significa n += 1 ? E' equivalente a scrivere n = n + 1

Classi e oggetti

Una classe è una struttura riusabile ed estensibile dotata di variabili specifiche (attributi) e funzioni specifiche (metodi).

La programmazione a oggetti è un modello di rappresentazione della realtà che permette di scrivere codice riutilizzabile e flessibile.

```
class Animale(nome, verso):
    ''' la mia prima classe '''
    def __init__(self, nome, verso):
        self.nome = nome
        self.verso = verso
        print 'E\' nato %s' % self.nome
    def verso():
        print self.verso
```

Eccezioni

"It's easier to ask forgiveness than it is to get permission"

Try, except, finally

Importare moduli

Import
From nome_modulo import nome

I moduli sono oggetti

Libreria standard

Cenni sulla libreria standard

Os e sys

Fare una richiesta HTTP con urllib2

Gestione delle dipendenze

Pip e easy_install

Cenni sull'introspezione

Programmazione riflessiva;
Ispezionare l'interno del codice in modo programmatico.

getattr
__docs__

Esempio reale di script

Clauz!

Libri consigliati

Dive into Python
Disponibile gratuitamente anche in italiano.
La versione del sito italiana al momento non funziona, io una copia.

Io ho letto: “Python from novice to professional”, molto completo ma è anche un bel mattone.

Il modo migliore per imparare

Followup

Scapy e Django

Donazioni e magliette

Fatevi sotto!

Maggiori Informazioni

Domande?

Altre info su

http://ninux.org http://blog.ninux.org/

http://map.ninux.org/

Contattaci a **contatti@ninux.org**

A cura di **Federico Capoano** **http://nemesisdesign.net/blog/**

Con il contributo di Claudio Pisa