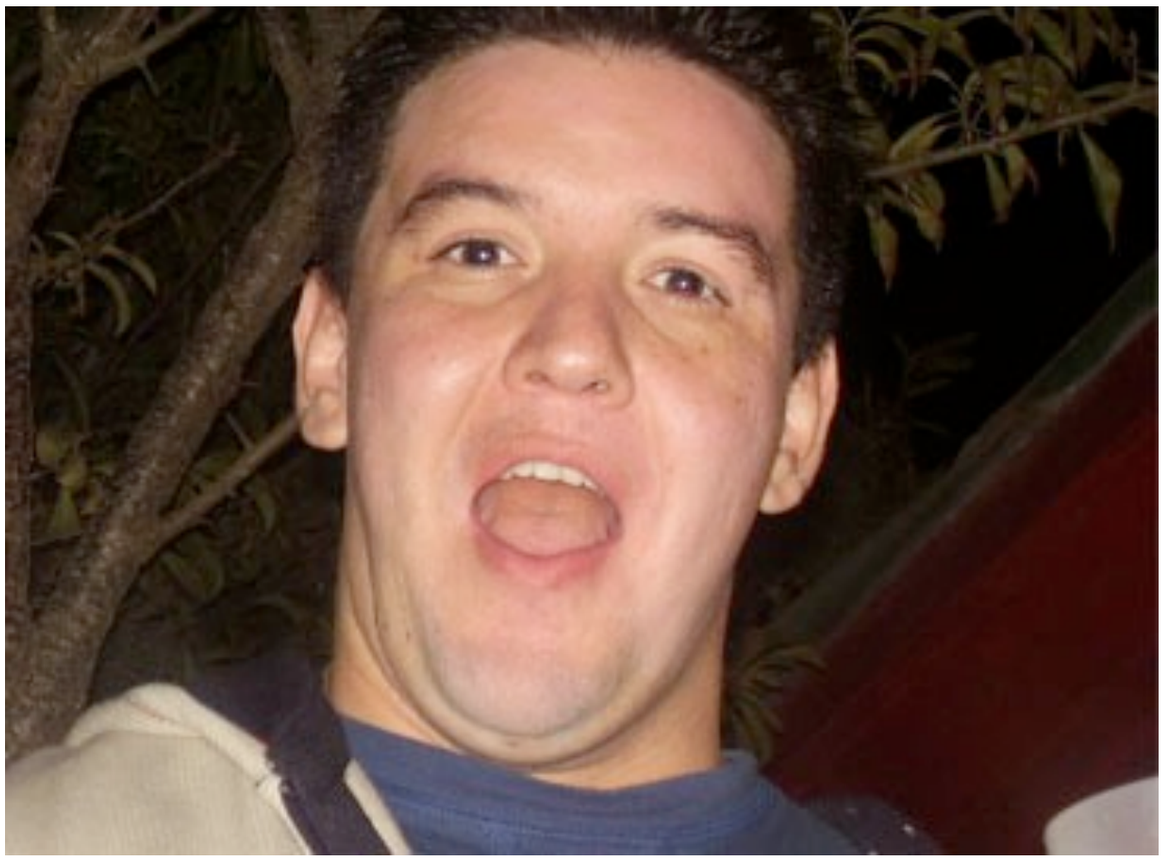
python™

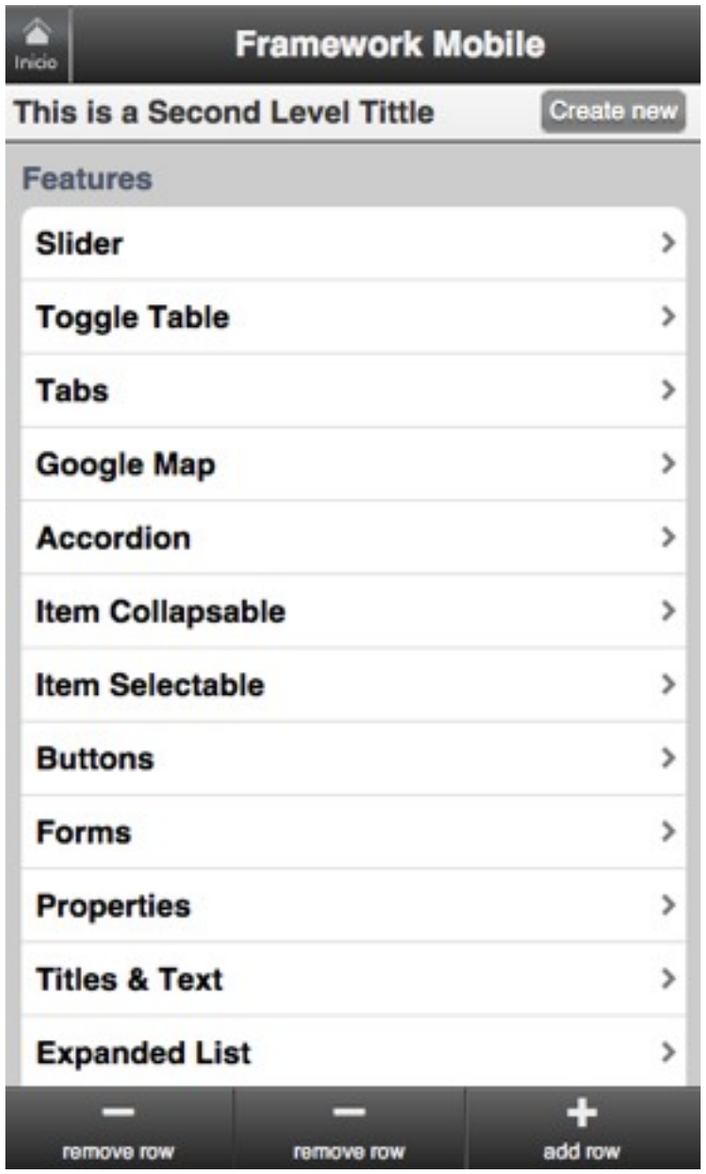
Avanzado
Parte I

El Charlista

@coto



@coto



Mobile Framework

JavaScript (2010)

Mobile Framework

JavaScript (2010)

STARTECHCONF

SANTIAGO CHILE 2011

Qompazz

INNOVACIÓN FINANCIERA


PYCON ARGENTINA
2012
12 AL 17 DE NOVIEMBRE
QUILMES - BUENOS AIRES

```
coto@cotos-pro ~
02:15 > python
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 01:25:11)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Iniciar Python

```
coto@cotos-pro ~
02:15 > python
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 01:25:11)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Los programas se ejecutan hasta que se alcanza EOF

- Con **Control-D** o **Control-Z** en el símbolo de sistema

- o escribe:
  `raise SystemExit`

Terminar Python

- Los programas se ejecutan hasta que se alcanza EOF

- Con **Control-D** o **Control-Z** en el símbolo de sistema

- o escribe:
  `raise SystemExit`

```
#!/usr/bin/env python
print "Hello World"
```

El truco del #!

```
#!/usr/bin/env python
print "Hello World"
```

El truco del #!

```
#!/usr/bin/env python
print "Hello World"
```

```
> python archivo.py
```

El truco del #!

```
#!/usr/bin/env python
print "Hello World"
```

El truco del #!

```
#!/usr/bin/env python
print "Hello World"
```

```
> chmod +x archivo.py
> ./archivo.py
```

```
>>> # suma
... 3+5
8

>>> # resta
... 3-4
-1

>>> # multiplicacion
... 2.0*3
6.0
```

Operadores

```
>>> # suma
... 3+5
8

>>> # resta
... 3-4
-1

>>> # multiplicacion
... 2.0*3
6.0
```

```
>>> # division
... 3/2
1.5

>>> # cociente
... 3//2
1

>>> # resto
... 3%2
1
```

Operadores

```
>>> # division
... 3/2
1.5

>>> # cociente
... 3//2
1

>>> # resto
... 3%2
1
```

```
>>> # potencia
... 3**2
9

>>> # left shift
... 5<<2
20

>>> # right shift
... 5>>2
1
```

Operadores

```
>>> # potencia
... 3**2
9

>>> # left shift
... 5<<2
20

>>> # right shift
... 5>>2
1
```

```
>>> # el operador ternario ?:
... ( 5 > 2 ) and “a” or “b”
```

Operadores

```
>>> # el operador ternario ?:
... ( 5 > 2 ) and “a” or “b”
```

Operadores

```
>>> # el operador ternario ?:
... ( 5 > 2 ) and “a” or “b”
“a”
```

Operadores

```
>>> # el operador ternario ?:
... ( 5 > 2 ) and “a” or “b”
“a”
```

```
>>> "a" if ( 5 > 2 ) else "b"
“a”
```

```
a = [2, 3, 4]                    # list de enteros
b = [2, 7, 3.5, "Hello"]         # lista mixta
c = []                           # lista vacia
d = [2, [a,b]]                   # Lista con otra lista
e = a + b                        # Unir dos listas
```

Manipulando Listas

```
x = a[1]                         # Obtén el 2º elemento
y = b[1:3]                       # Obtén una sublista
z = d[1][0][2]                   # Listas anidadas
b[0] = 42                        # Cambiar un elemento
```

Listas

```
a = [2, 3, 4]                    # list de enteros
b = [2, 7, 3.5, "Hello"]         # lista mixta
c = []                           # lista vacia
d = [2, [a,b]]                   # Lista con otra lista
e = a + b                        # Unir dos listas
```

Manipulando Listas

```
x = a[1]                         # Obtén el 2º elemento
y = b[1:3]                       # Obtén una sublista
z = d[1][0][2]                   # Listas anidadas
b[0] = 42                        # Cambiar un elemento
```

```
f = (2,3,4,5)                          # tupla de enteros
g = (,)                                # tupla vacia
h = (2, [3,4], (10,11,12))             # tupla mixta
```

Manipulando Tuplas

```
x = f[1]                               # Obtén el 2º elemento
y = f[1:3]                             # Obtén una porción
z = h[1][1]                            # Agrupamiento
```

Tuplas

```
f = (2,3,4,5)                          # tupla de enteros
g = (,)                                # tupla vacia
h = (2, [3,4], (10,11,12))             # tupla mixta
```

Manipulando Tuplas

```
x = f[1]                               # Obtén el 2º elemento
y = f[1:3]                             # Obtén una porción
z = h[1][1]                            # Agrupamiento
```

Tuplas

```
f = (2,3,4,5)                          # tupla de enteros
g = (,)                                # tupla vacia
h = (2, [3,4], (10,11,12))             # tupla mixta
```

Manipulando Tuplas

```
x = f[1]                               # Obtén el 2º elemento
y = f[1:3]                             # Obtén una porción
z = h[1][1]                            # Agrupamiento
```

- Las tuplas son como las listas, pero con tamaño definido.
- No se pueden reemplazar miembros.

```
a = { }
b = { 'x': 3, 'y': 4 }
c = { 'uid': 105,
      'login': 'beazley',
      'name' : 'David Beazley'
    }
```

Acceso a diccionarios

```
u = c['uid']                        # Acceso a un elemento
c['shell'] = "/bin/sh"              # Crear elemento
"name" in c                         # Chequear elemento
c.get("name", "no value")           # Chequear elemento
```

Diccionarios

```
a = { }
b = { 'x': 3, 'y': 4 }
c = { 'uid': 105,
      'login': 'beazley',
      'name' : 'David Beazley'
    }
```

Acceso a diccionarios

```
u = c['uid']                        # Acceso a un elemento
c['shell'] = "/bin/sh"              # Crear elemento
"name" in c                         # Chequear elemento
c.get("name", "no value")           # Chequear elemento
```

```
lambda arguments: expression
```

```
def name(arguments):
    return expression
```

```
> f = lambda x, y : x + y
> f(1,1)
2
```

Lambdas y Func. Prog.

```
lambda arguments: expression
```

```
def name(arguments):
    return expression
```

```
> f = lambda x, y : x + y
> f(1,1)
2
```

```
> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
> print map(lambda x: x * 2 + 10, foo)
...[14, 46, 28, 54, 44, 58, 26, 34, 64]
```

```
> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print filter(lambda x: x % 3 == 0, foo)
...[18, 9, 24, 12, 27]
```

```
> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print reduce(lambda x, y: x + y, foo)
...139
```

Lambdas y Func. Prog.

```
> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
> print map(lambda x: x * 2 + 10, foo)
...[14, 46, 28, 54, 44, 58, 26, 34, 64]
```

```
> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print filter(lambda x: x % 3 == 0, foo)
...[18, 9, 24, 12, 27]
```

```
> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print reduce(lambda x, y: x + y, foo)
...139
```

```
#!/usr/bin/env python
def makebold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def makeitalic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

@makebold
@makeitalic
def hello():
    return "hello world"

print hello() ## returns <b><i>hello world</i></b>
```

Decorators

```
#!/usr/bin/env python
def makebold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def makeitalic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

@makebold
@makeitalic
def hello():
    return "hello world"

print hello() ## returns <b><i>hello world</i></b>
```

```
class Account:
    def __init__(self, initial):
        self.balance = initial
    def deposit(self, amt):
        self.balance = self.balance + amt
    def withdraw(self,amt):
        self.balance = self.balance - amt
    def getbalance(self):
        return self.balance
```

Classes

```
class Account:
    def __init__(self, initial):
        self.balance = initial
    def deposit(self, amt):
        self.balance = self.balance + amt
    def withdraw(self,amt):
        self.balance = self.balance - amt
    def getbalance(self):
        return self.balance
```

```
a = Account(1000.00)
a.deposit(550.23)
a.deposit(100)
a.withdraw(50)
print a.getbalance()
```

Classes

```
a = Account(1000.00)
a.deposit(550.23)
a.deposit(100)
a.withdraw(50)
print a.getbalance()
```

Classes

```
a = Acco
a.deposi
a.deposi
a.withdr
print a.
```

```
class Electricity(object):
        """Describe the electricity"""
        def __init__(self):
            # the self variable represents the instance of the object itself.
            # Constructor for instances, it will overwrite Class attributes
            # when it is called
            self._private = 'inside'
            self._voltage = 220

        @property
        def voltage(self):
            return self._voltage

        @voltage.setter
        def voltage(self, value):
            self._voltage = value/2

        _private = 'outside'
        _another_private = 'boo'

        @voltage.deleter
        def voltage(self):
            del self._voltage

        @staticmethod
        def metodo_estatico(val1, val2):
            return "Hello %s and %s" % (val1, val2)

        @classmethod
        def metodo_class(cls, val2):
            return "Hello %s and %s" % (cls, val2)
```

Classes

```
class Electricity(object):
        """Describe the electricity"""
        def __init__(self):
            # the self variable represents the instance of the object itself.
            # Constructor for instances, it will overwrite Class attributes
            # when it is called
            self._private = 'inside'
            self._voltage = 220

        @property
        def voltage(self):
            return self._voltage

        @voltage.setter
        def voltage(self, value):
            self._voltage = value/2

        _private = 'outside'
        _another_private = 'boo'

        @voltage.deleter
        def voltage(self):
            del self._voltage

        @staticmethod
        def metodo_estatico(val1, val2):
            return "Hello %s and %s" % (val1, val2)

        @classmethod
        def metodo_class(cls, val2):
            return "Hello %s and %s" % (cls, val2)
```

```
try:
    f = open("foo")
except IOError, e:
    print "Couldn't open 'foo'. Error: {}".format(e)
```

```
try:
    f = open("foo")
except Exception:
    print "Couldn't open 'foo'. Sorry."
```

Exceptions

```
try:
     f = open("foo")
except IOError, e:
     print "Couldn't open 'foo'. Error: {}".format(e)
```

```
try:
     f = open("foo")
except Exception:
     print "Couldn't open 'foo'. Sorry."
```

```
import re
r = re.compile(r'(\d+)\.(\d*)')
m = r.match("42.37")
a = m.group(0)          # Returns '42.37'
b = m.group(1)          # Returns '42'
c = m.group(2)          # Returns '37'
print m.start(2)        # Prints 3
```

Expresiones Regulares

```
import re
r = re.compile(r'(\d+)\.(\d*)')
m = r.match("42.37")
a = m.group(0)          # Returns '42.37'
b = m.group(1)          # Returns '42'
c = m.group(2)          # Returns '37'
print m.start(2)        # Prints 3
```

- Seguridad en Python
- Interfaces de Sistema Operativo
- Trabajando con Threads
- Programando en Red
- Interfaces de Base de Datos
- Ejecución restringida
- Extensiones en C

Próximamente...

- Seguridad en Python
- Interfaces de Sistema Operativo
- Trabajando con Threads
- Programando en Red
- Interfaces de Base de Datos
- Ejecución restringida
- Extensiones en C

Gracias!!

@coto