Quick Intro to Python

Steven Clontz

COMP 7970
Auburn University

February ??, 2013

Table of Contents

Abstract

- Presentation available at https://github.com/StevenClontz/2013-python-presentation
- I'll talk about basic syntax, data types, loops, functions, conditionals etc. in Python.
- It should be enough to get anyone started hacking, at least.

Abstract

- Presentation available at https://github.com/StevenClontz/2013-python-presentation
- I'll talk about basic syntax, data types, loops, functions, conditionals etc. in Python.
- It should be enough to get anyone started hacking, at least.

Abstract

- Presentation available at https://github.com/StevenClontz/2013-python-presentation
- I'll talk about basic syntax, data types, loops, functions, conditionals etc. in Python.
- It should be enough to get anyone started hacking, at least.

Getting Started

- Python's syntax encourages beautiful code, using whitespace to organize rather than combinations of braces and semicolons.
- We also don't worry about declaring variables or defining complex data types - most of what you'll need comes in the box. If it's not there, we can always import it.

Getting Started

- Python's syntax encourages beautiful code, using whitespace to organize rather than combinations of braces and semicolons.
- We also don't worry about declaring variables or defining complex data types - most of what you'll need comes in the box. If it's not there, we can always import it.

Getting Started

- Python's syntax encourages beautiful code, using whitespace to organize rather than combinations of braces and semicolons.
- We also don't worry about declaring variables or defining complex data types - most of what you'll need comes in the box. If it's not there, we can always import it.

C and Python:

```
void foo(int x)
{if (x == 0) {

    bar();baz();

  } else {
    qux(x);
    foo(x - 1);}
}
```

```
def foo(x):
  if x == 0:
    bar()
    baz()
  else:
    qux(x)
    foo(x - 1)
```

Data Types

- Strings:

```
name = "Stevie"
hello = 'Hi %s' % name  # or 'Hi $name'
blank = ""  # or str()
```

- Numbers:

```
three = 1 + 2  # an integer
one_point_two_ish = -1 + 2.2  # a float
degree_135 = -2 + 2j  # gaussian integer
```

  - Big numbers? No problem!

```
small = 3
big = small**small**small
  # 3^(3^3) = 7625597484987
```

Data Types

- Strings:

```
name = "Stevie"
hello = 'Hi %s' % name  # or 'Hi $name'
blank = ""  # or str()
```

- Numbers:

```
three = 1 + 2  # an integer
one_point_two_ish = -1 + 2.2  # a float
degree_135 = -2 + 2j  # gaussian integer
```

  - Big numbers? No problem!

```
small = 3
big = small**small**small
  # 3^(3^3) = 7625597484987
```

Data Types

- Strings:

```
name = "Stevie"
hello = 'Hi %s' % name  # or 'Hi $name'
blank = ""  # or str()
```

- Numbers:

```
three = 1 + 2  # an integer
one_point_two_ish = -1 + 2.2  # a float
degree_135 = -2 + 2j  # gaussian integer
```

  - Big numbers? No problem!

```
small = 3
big = small**small**small
  # 3^(3^3) = 7625597484987
```

Data Types

Iterables:

- Lists:

```
less_than_four = [0, 1, 2, 3]  # or range(4)
two_words = list().append("Hi").append("Mom")
comprende = [2**n for n in less_than_four]
  # [1, 2, 4, 8]
```

- Tuples:

```
five_away = (3, 4)
five_away.append(1) # ERROR!
```

Data Types

Iterables:

- Lists:

```
less_than_four = [0, 1, 2, 3]  # or range(4)
two_words = list().append("Hi").append("Mom")
comprende = [2**n for n in less_than_four]
  # [1, 2, 4, 8]
```

- Tuples:

```
five_away = (3, 4)
five_away.append(1) # ERROR!
```

Data Types

Iterables:

- Lists:

```
less_than_four = [0, 1, 2, 3]  # or range(4)
two_words = list().append("Hi").append("Mom")
comprende = [2**n for n in less_than_four]
  # [1, 2, 4, 8]
```

- Tuples:

```
five_away = (3, 4)
five_away.append(1) # ERROR!
```

Data Types

More iterables:

- Dictionaries:

```
named = {
   "one": 1,
   "two comma three": (2, 3),
   4: "four (backwards)"
}

   # named['one'] == 1
emptydict = dict()  # or {}
```

- Sets:

```
no_dupes = set([1, 2, 1, "two"])
   # or {1, 2, "two"}
emptyset = set()  # not {}
```

Data Types

More iterables:

- Dictionaries:

```
named = {
   "one": 1,
   "two comma three": (2, 3),
   4: "four (backwards)"
}

   # named['one'] == 1
emptydict = dict()   # or {}
```

- Sets:

```
no_dupes = set([1, 2, 1, "two"])
   # or {1, 2, "two"}
emptyset = set()   # not {}
```

Operators

```
3 == 2 + 1
3 == 1.5 * 2
3 == 0.6 * 5
3 != "three"
True and (1 == 3 - 2)
some_int == (some_int / 3) * 3 + some_int % 3
  # Warning: Python 3 requires some_int // 3
```

Functions

```
def list_m_powers(x, m):
    n = 1
    powers = []
    while n < m:
        powers.append(x**n)
        n += 1
    return powers

simple = lambda x, m: [x**n for n in range(1, m)]

    # list_m_powers(2, 5) == simple(2, 5)
    # == [2, 4, 8, 16, 32]
```

Functions

```
def list_m_powers(x, m):
  n = 1
  powers = []
  while n < m:
    powers.append(x**n)
    n += 1
  return powers

simple = lambda x, m: [x**n for n in range(1, m)]

  # list_m_powers(2, 5) == simple(2, 5)
  # == [2, 4, 8, 16, 32]
```

Loops, Conditionals

```
def do_stuff(items):
  copy = items[:]
  while copy != []:
    item = copy.pop()
    for index, numb in enumerate(item):
      print numb**index
    if len(item) < 3:
      print "That was easy!\n"
    else:
      print "Whew, now we're done!\n"
```

```
# do_stuff([[1, 2], [3, 5, 7, -4]]) returns...
1
5
49
-64
Whew, now we're done!

1
2
That was easy!
```

That's all folks.