

Python Tutorial

Benjamin L. Zaitlen

Biocomplexity Institute, Indiana University

August 15, 2007

- 1 Introduction
- 2 Interactive Shell
- 3 Strings and Numbers
- 4 Storing Data
- 5 Control Structures
- 6 Functions
- 7 Classes
- 8 CompuCell Classes



Introduction

- Created Guido van Rossum in 1991
- Human-readable
- Fast creation and debugging
- See <http://www.python.org> and Python Essential Reference by David Beazley



Introduction

- Created Guido van Rossum in 1991
- Human-readable
- Fast creation and debugging
- See <http://www.python.org> and Python Essential Reference by David Beazley



Introduction

- Created Guido van Rossum in 1991
- Human-readable
- Fast creation and debugging
- See <http://www.python.org> and Python Essential Reference by David Beazley



Introduction

- Created Guido van Rossum in 1991
- Human-readable
- Fast creation and debugging
- See <http://www.python.org> and Python Essential Reference by David Beazley



Comments and Quirks

- # A comment
- Python is space sensitive. Lines after a : must be indented.



Comments and Quirks

- # A comment
- Python is space sensitive. Lines after a : must be indented.



Comments and Quirks

- # A comment
- Python is space sensitive. Lines after a : must be indented.

Indentation Example

```
for i in (1,2,3,4):  
    print i,
```

```
1 2 3 4
```



The Shell

- Invoke the shell from the command line using the keyword `python`
- Useful for basic math, testing ideas
- Do not build programs in interpreter



The Shell

- Invoke the shell from the command line using the keyword `python`
- Useful for basic math, testing ideas
- Do not build programs in interpreter



The Shell

- Invoke the shell from the command line using the keyword `python`
- Useful for basic math, testing ideas
- Do not build programs in interpreter



The Shell

- Invoke the shell from the command line using the keyword python
- Useful for basic math, testing ideas
- Do not build programs in interpreter

Interpreter

```
>>>print "Hello, World!"  
Hello, World!  
>>>var = 9+2  
>>>var*11  
121
```



Strings

- A simple string "hello world"
- Concatenation: "hello"+" world" → "hello world"
- Repetition: "hello"*3 → "hellohellohello"
- Indexing: "world"[3] → "l"
 - Note: python lists are zero-offset
- Searching: "o" in "hello" → True



Strings

- A simple string "hello world"
- Concatenation: "hello"+" world" → "hello world"
- Repetition: "hello"*3 → "hellohellohello"
- Indexing: "world"[3] → "l"
 - Note: python lists are zero-offset
- Searching: "o" in "hello" → True



Strings

- A simple string "hello world"
- Concatenation: "hello"+" world" → "hello world"
- Repetition: "hello"*3 → "hellohellohello"
- Indexing: "world"[3] → "l"
 - Note: python lists are zero-offset
- Searching: "o" in "hello" → True



Strings

- A simple string "hello world"
- Concatenation: "hello"+" world" → "hello world"
- Repetition: "hello"*3 → "hellohellohello"
- Indexing: "world"[3] → "l"
 - Note: python lists are zero-offset
- Searching: "o" in "hello" → True



Strings

- A simple string "hello world"
- Concatenation: "hello"+" world" → "hello world"
- Repetition: "hello"*3 → "hellohellohello"
- Indexing: "world"[3] → "l"
 - Note: python lists are zero-offset
- Searching: "o" in "hello" → True



Strings

- A simple string "hello world"
- Concatenation: "hello"+" world" → "hello world"
- Repetition: "hello"*3 → "hellohellohello"
- Indexing: "world"[3] → "l"
 - Note: python lists are zero-offset
- Searching: "o" in "hello" → True



Numbers

- Basic math notation: 1.4, 2+2, 2**10, 1e10
 - Note: Integer division floors values: $2/3 \rightarrow 0$, $2./3 \rightarrow .6667$
- Math functions require import of math
 - `import math`
 - `math.sqrt(4) → 2.0`
 - `from math import *`
 - `sqrt(4) → 2.0`



Numbers

- Basic math notation: 1.4 , $2+2$, $2^{**}10$, $1e10$
 - Note: Integer division floors values: $2/3 \rightarrow 0$, $2./3 \rightarrow .6667$
- Math functions require import of math
 - `import math`
 - `math.sqrt(4) → 2.0`
 - `from math import *`
 - `sqrt(4) → 2.0`



Numbers

- Basic math notation: 1.4 , $2+2$, $2^{**}10$, $1e10$
 - Note: Integer division floors values: $2/3 \rightarrow 0$, $2./3 \rightarrow .6667$
- Math functions require import of math
 - `import math`
 - `math.sqrt(4) → 2.0`
 - `from math import *`
 - `sqrt(4) → 2.0`



Numbers

- Basic math notation: 1.4 , $2+2$, $2^{**}10$, $1e10$
 - Note: Integer division floors values: $2/3 \rightarrow 0$, $2./3 \rightarrow .6667$
- Math functions require import of math
 - `import math`
 - `math.sqrt(4) → 2.0`
 - `from math import *`
 - `sqrt(4) → 2.0`



Numbers

- Basic math notation: 1.4 , $2+2$, $2^{**}10$, $1e10$
 - Note: Integer division floors values: $2/3 \rightarrow 0$, $2./3 \rightarrow .6667$
- Math functions require import of math
 - `import math`
 - `math.sqrt(4) → 2.0`
 - `from math import *`
 - `sqrt(4) → 2.0`



Numbers

- Basic math notation: 1.4 , $2+2$, $2^{**}10$, $1e10$
 - Note: Integer division floors values: $2/3 \rightarrow 0$, $2./3 \rightarrow .6667$
- Math functions require import of math
 - `import math`
 - `math.sqrt(4) \rightarrow 2.0`
 - `from math import *`
 - `sqrt(4) \rightarrow 2.0`



Numbers

- Basic math notation: 1.4 , $2+2$, $2^{**}10$, $1e10$
 - Note: Integer division floors values: $2/3 \rightarrow 0$, $2./3 \rightarrow .6667$
- Math functions require import of math
 - `import math`
 - `math.sqrt(4) \rightarrow 2.0`
 - `from math import *`
 - `sqrt(4) \rightarrow 2.0`



Variables and List

Dynamically-Typed Variables

- `x = 5`
- `x = 3.14`
- `x = 'text'`
- `x = '3.14'`

Dynamically-Typed Lists

- `numbers = [0,1,2,3,4,5]`
- `words = ['compucell','workshop']`
- `combo = [12,23,['text','knot']] + words`



List Operations

- `words.append('almond')` → `['compucell','workshop','almond']`
- `words.insert(1,'water')` → `['compucell','water','workshop','almond']`
- `words.reverse()` → `['almond','workshop','water','compucell']`
- `words.remove('water')` → `['almond','workshop','compucell']`



Dictionaries aka Hash Tables, Associative Arrays, Lookup Tables

- `dictionary = {'indefatigable':'untiring',
'intrepid':'fearlessness','dissemble':'simulate'}`
- `constants = {'pi':3.1415, 'e':2.7182, 'phi':1.6180}`
- `com_dict = {1:[1,2,3],2:[1,0,3],3:[0,4,5]}`



Dictionaries Operations

- `com_dict.keys()` → [1,2,3]
- `com_dict.values()` → [[1, 2, 3], [1, 0, 3], [0, 4, 5]]

Accessing Members

- `com_dict[3]` → [0,4,5]
- `constants['phi']` → 1.6180



If, While, and For

The If...

```
if condition:
    statements
elif condition:
    statements
else condition:
    statements
```

Note:Beware of Python's
requirement for indentation

The While...

```
while condition:
    statements
```

The For...

```
for var in sequence:
    statements
```



Control Structure Examples

The If...

```
if (x > 0):  
    print "Positive"  
elif (x < 0):  
    print "Negative"  
else:  
    Print "Zero"
```

The While...

```
while (1):  
    print "Always true"
```

The For...

```
for i in range(5):  
    print i
```



Defining Functions

No Arguments

```
def Hello_World():  
    print 'Hello World'  
def return_5():  
    return 5
```

Note: Python passes all arguments by reference. This means changing the value in the function will change the value when the program exits the function

With Arguments

You can pass a single variable:

```
def one_var(x):  
    print 'You passed the variable: %s' % (x)
```

You can also pass a list:

```
def list_func(list):  
    for i in list:
```



Calling Functions

No Arguments

```
>>>Hello_World()
Hello World
>>>return_5()
5
```

With Arguments

Passing a single variable:

```
>>>x = 'my variable'
>>>one_var(x)
my variable
```

Passing a list:

```
list = [0,1,2,3,4]
>>>list_func(list):
0 1 2 3 4
```

In General

```
def name(arg1,arg2,...):
    statements
    return expression
```



The Container for Everything

What is a Class ?

- Classes contain stuff. What kind of stuff ?
Anything!
- What are they good for ? **Almost everything**

What can they store ?

Classes can store everything: variables, lists, dictionaries, functions, other lists, etc



Typical Python Class

Defining

```
class Complex:
    var_every = 0          #class variable
    def __init__(self, var = 0):
        self.instance = var    #instance variable
    def class_print(self):
        print "My Value is: %s" %(self.instance)
```



Classes Continued...

General Class

```
class name(baseclass1, baseclass2, ...):  
    statements  
    def name(self, arg1, arg2, ...):  
        more statements
```

Example: The Complex Variables

One object contains both the real and imaginary parts of a complex variable. Can be achieved with a list, dictionary, or class. Class seems to make the most sense...



The Complex Class

Defining

```
class Complex:
    def __init__(self, real, imag):
        self.r = real
        self.i = imag
    def cprint(self):
        if(self.i < 0):
            print "%.3f%.3f*i" %(self.r, self.i)
        else:
            print "%.3f+%.3f*i" %(self.r, self.i)
```

- **self** is like **this** and must be passed to member functions



Complex Class Continued...

Initializing

- Let's initialize $1-i$



Complex Class Continued...

Initializing

- Let's initialize $1-i$
- `Complex(1,-1)`



Complex Class Continued...

Initializing

- Let's initialize $1-i$
- `Complex(1,-1)`
- We need to store the class. Simple:
`comp = Complex(1,-1)`. Now we have access to the variables `r` and `i` and the member function `cprint()`



Complex Class Continued...

Initializing

- Let's initialize $1-i$
- `Complex(1,-1)`
- We need to store the class. Simple:
`comp = Complex(1,-1)`. Now we have access to the variables `r` and `i` and the member function `cprint()`

Accessing Members

- Use the `.` operator to access variables
`comp.r` `comp.i`

Complex Class Continued...

Initializing

- Let's initialize $1-i$
- `Complex(1,-1)`
- We need to store the class. Simple:
`comp = Complex(1,-1)`. Now we have access to the variables `r` and `i` and the member function `cprint()`

Accessing Members

- Use the `.` operator to access variables
`comp.r` `comp.i`
- Use the `.` operator to access functions
`comp.cprint()`

The Cell

What are the attributes of a cell ?

- Type



The Cell

What are the attributes of a cell ?

- Type
- Volume



The Cell

What are the attributes of a cell ?

- Type
- Volume
- Center of Mass

These are a few of the variables inside the cell class built for CompuCell. We can interface with this class through Python

