Formal Methods II: Brief intro to Python

K. Dermitzakis
11.10.2013

Heavily based on presentation given by Nico Schmidt (AI Lab) and slides from Harvard’s telescope data center (TDC)
Python, current version 3.3.2

• open source
• general purpose, high-level programming language
• philosophy: easy, intuitive coding, readability
• comes with large standard library
• object oriented, procedural, functional
• scripting or executables
• dynamic type system
Interactive shell

useful for:
• learning python
• playing around with python and the libs
• testing your own modules

other python-shells:
• ipython (http://ipython.org)
• IDLE (written in python with GUI-toolkit Tkinter)
• Pythonxy (Qt and Spyder based)
Eclipse - PyDev

• Eclipse IDE integration
• highlighting, tab-completion, shows errors/warnings while typing
• useful for larger projects/ programs with multiple source files
• easy to debug your code
Modules

NumPy:
- scientific computing with python
- sophisticated array facility (matrix algebra)
- numeric linear algebra algorithms (QR-decomposition, Eigen value-decomposition,...)
- random number capabilities

Matplotlib:
- plotting library
- generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc
- similar to Matlab plotting functions
Installation

Python: [http://python.org/download/](http://python.org/download/)

IPython: [http://ipython.org/download.html](http://ipython.org/download.html)

PyDev: [http://pydev.org/download.html](http://pydev.org/download.html)


Documentation

http://python.org/doc/:

- Standard library reference: http://docs.python.org/library/
- Language Reference http://docs.python.org/reference/
- Grammar http://docs.python.org/reference/grammar.html
A code example

```python
x = 34 - 23                   # A comment.
y = "Hello"                   # Another one.
z = 3.45

if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"            # String concat.

print(x)
print(y)
```
The basics

• Assignment uses = and comparison uses ==.
• For numbers + - * / % are as expected.
  – Special use of + for string concatenation.
  – Special use of % for string formatting (as with printf in C)
• Logical operators are words (and, or, not) not symbols
• The basic printing command is print.
• The first assignment to a variable creates it.
  – Variable types don’t need to be declared, variable types are automatically chosen by Python on assignment.
Basic datatypes

- **Integers** (default for numbers)
  \[ z = 5 / 2 \]  # Answer is 2, integer division.

- **Floats**
  \[ x = 3.456 \]

- **Strings**
  - Can use "" or ‘’ to specify.
    “abc” ‘abc’ (Same thing.)
  - Unmatched can occur within the string.
    “matt’s”
  - Use triple double-quotes for multi-line strings or strings than contain both ‘and “ inside of them:
    """"a‘b“c""""

Whitespace and indentation

Whitespace is meaningful in Python: especially for indentation and placement of newlines.

- **Use a newline to end a line of code.**
  - Use \ when must go to next line prematurely.

- **No braces `{}` to mark blocks of code in Python… Use consistent indentation instead.**
  - The first line with *less* indentation is outside of the block.
  - The first line with *more* indentation starts a nested block.

- **Often a colon appears at the start of a new block.** (E.g. for function and class definitions.)
Comments

• Start comments with `#` – the rest of line is ignored.
• Can include a “documentation string” as the first line of any new function or class that you define.
• The development environment, debugger, and other tools use it: it’s good style to include one.

```python
def my_function(x, y):
    """This is the docstring. This function does blah blah blah blah."""
    # The code would go here...
```
Variable assignment

• *Binding a variable* in Python means setting a *name* to hold a *reference* to some *object*.
  – Assignment creates references, not copies

• *Names in Python do not have an intrinsic type. Objects have types.*
  – Python determines the type of the reference automatically based on the data object assigned to it.

• *You create a name the first time it appears on the left side of an assignment expression:*
  \[
  x = 3
  \]

• *A reference is deleted via garbage collection after any names bound to it have passed out of scope.*

• *Multiple Assignment*
  – You can also assign to multiple names at the same time.
  \[
  x, y = 2, 3
  \]
Naming rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.
  
  bob  Bob  _bob  _2_bob_  bob_2  BoB

- There are some reserved words:

  and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while
Flow control examples

```python
if x == 3:
    print(“X equals 3.”)
elif x == 2:
    print(“X equals 2.”)
else:
    print(“X equals something else.”)
print(“This is outside the ‘if’.”)

assert(number_of_players < 5)

x = 3
while x < 10:
    if x > 7:
        x += 2
        continue
    x = x + 1
print(“Still in the loop.”)
if x == 8:
    break
print(“Outside of the loop.”)

for x in range(10):
    if x > 7:
        x += 2
        continue
    x = x + 1
print(“Still in the loop.”)
if x == 8:
    break
print(“Outside of the loop.”)
```
Functions

- `def` creates a function and assigns it a name
- `return` sends a result back to the caller
- Arguments are passed by assignment
- Arguments and return types are not declared

```python
def <name>(arg1, arg2, ..., argN):
    <statements>
    return <value>

def times(x, y):
    return x*y
```
Gotchas

• All functions in Python have a return value
  – even if no return line inside the code.
• Functions without a `return` return the special value `None`.
• There is no function overloading in Python.
  – Two different functions can’t have the same name, even if they have different arguments.
• Functions can be used as any other data type. They can be:
  – Arguments to function
  – Return values of functions
  – Assigned to variables
  – Parts of tuples, lists, etc
Tutorials
Reference semantics

- Assignment manipulates references
  - \( x = y \) does not make a copy of the object \( y \) references
  - \( x = y \) makes \( x \) reference the object \( y \) references

- Very useful; but beware!
- Example:
  
  ```python
  a = [1, 2, 3]  # a now references the list [1, 2, 3]
b = a          # b now references what a references
a.append(4)    # this changes the list a references
print(b)       # if we print what b references
  ```

What is the value of \( b \)??
There is a lot going on when we type:
\[ x = 3 \]
- First, an integer 3 is created and stored in memory
- A name \( x \) is created
- A reference to the memory location storing the 3 is then assigned to the name \( x \)
- When we say that the value of \( x \) is 3 we mean that \( x \) now refers to the integer 3
Mutable and immutable types

• The data 3 we created is of type integer. In Python, the datatypes integer, float, and string (and tuple) are “immutable.”
• This doesn’t mean we can’t change the value of x, i.e. *change what x refers to* ...
• For example, we could increment x:

```python
x = 3
y = x
y = 4
print(x)
```
What is the value of x?

• For other data types (lists, dictionaries, user-defined types), assignment works differently.
  – These datatypes are “mutable.”
  – When we change these data, we do it *in place.*
  – We don’t copy them into a new memory address each time.
  – If we type `y=x` and then modify y, both x and y are changed.
Passing arguments to functions

- Arguments are passed by *assignment*
- Passed arguments are assigned to *local names*
- Assignment to argument names don't affect the caller
- Changing a mutable argument may affect the caller
  
  ```python
  def changer(x,y):
      x = 2  # changes local value of x only
      y[0] = 'hi'  # changes shared object
  ```

- Can define defaults for arguments that need not be passed (optional arguments)
  
  ```python
  def func(a, b, c=10, d=100):
      print(a, b, c, d)
  ```