Introduction to Python

A quick tour of the environment and programming language

Dominique Gerald M Cimafranca | blog: villageidiotsavant.com
Ateneo de Davao University | email: dominique@cimafranca.com

Reference Materials

- Official web site
  - www.python.org
- Official Documentation
  - doc.python.org
- Free textbooks
  - *Think Python: How to Think Like a Computer Scientist* by Allen Downey (http://www.greenteapress.com/thinkpython/)
  - *Learnng with Python* by Jeffrey Elkner, Allen Downey, and Chris Meyers
    (http://www.openbookproject.net/thinkcs/python/english2e/index.html)

What is Python?

- Easy-to-learn, powerful programming language
- Efficient high-level data structures
- Simple approach to object-oriented programming
- Elegant syntax
- Dynamic typing
- Well-suited for scripting and rapid application dev't.
- ...and free!

Python environment

- Components
  - Interpreter
  - Standard library
  - Modules
- Available for almost all platforms
  - Windows
  - Linux
  - Unix
- Implementations
  - CPython
  - Jython
  - Iron Python
- IDEs
  - IDLE
  - Geany
  - Eclipse

Let's run Python!

Our working environment

- Windows XP / Windows 7
- CPython reference implementation
- IDLE IDE
  - Interpreter Console
  - Scripts

Python Interactive Mode

- Executes commands directly
- Processes expressions and returns results
- Error handling

A sample Python program

```
import random

mynumber=int(random.random()*100)+1
guess=None

print "I'm thinking of a number between 1 and 100."

while(guess != mynumber):
    guess=eval(raw_input("Guess my number? "))
    if guess > mynumber:
        print "Too high! Try again!"
    elif guess < mynumber:
        print "Too low! Try again!"
    else:
        print "You got it!"
```

What do you notice about Python?

A sample Python program

```
import random

mynumber=int(random.random()*100)+1
guess=None

print "I'm thinking of a number between 1 and 100."

while(guess != mynumber):
    guess=eval(raw_input("Guess my number? "))
    if guess > mynumber:
        print "Too high! Try again!"
    elif guess < mynumber:
        print "Too low! Try again!"
    else:
        print "You got it!"
```

Dissecting our sample

```
import random

mynumber=int(random.random()*100)+1
guess=None

print "I'm thinking of a number between 1 and 100."

while(guess != mynumber):
    guess=eval(raw_input("Guess my number? "))
    if guess > mynumber:
        print "Too high! Try again!"
    elif guess < mynumber:
        print "Too low! Try again!"
    else:
        print "You got it!"
```

Load an external library

Variable assignment

Loop

Conditional

Indentations to delineate code blocks

Python as a Calculator

- Numeric Data Types
  - `int`
  - `float`
  - `long`
  - `complex`
- Built-in Operands
  - Arithmetic: **+ - * / ** %**
  - Comparison: **== != < <= > >=**
  - Bitwise: **& | ^ ~ << >>**
- Data type conversion
- Additional functions from math and cmath libraries

Variables

- Variables are dynamically-typed
  - No need to declare
  - Change type along with values assigned
  - But: error if used before assignment
- Simultaneous assignment
  - `x = y = z = 0`
- Positional assignment
  - `a, b, c = 2, 3, 4`

Strings

```
'hello world'

'doesn\'t'

“doesn't”

' “Yes,” he said. '

message = 'this is quite a long \
message and so we must break the \
lines with slashes.'
```

String operations

- Concatenation
  - `'hello' + ' world'`
- Repetition
  - `'hello' * 5`
- Conversion
  - `str(x)` - convert number x to a string
  - `int(s)` - convert string s to an integer
  - Likewise for `float()` and `long()`
- Length
  - `len('hello')` → `5`

String subscripts and slices

```
msg = 'Adventure Time'

msg[0]  → 'A'
msg[4]  → 'n'

msg[-1] → 'e'
msg[-3] → 'i'

msg[0:9] → 'Adventure'
msg[2:-5] → 'venture'

msg[:2] → 'Ad'
```

Built-in string methods

- `str.capitalize()`
- `str.center(width[, fillcar])`
- `str.count(sub[, start[, end]])`
- `str.endswith(suffix[, start[, end])`
- `str.expandtabs([tabsize])`
- `str.find(sub[, start[, end]])`
- `str.index(sub[, start[, end]])`
- `str.isalnum()`
- `str.isalpha()`
- `str.islower()`
- `str.isspace()`

...and many, many more!

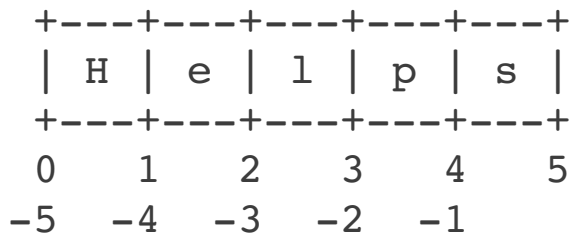
(see http://docs.python.org/library/stdtypes.html)

Interlude: Exploring the Python environment with dir() and help()

Python does not have arrays.

Notes about strings

- There is no character data type in Python
- Strings are immutable
- Other string variations
  - raw
  - unicode
- Think of slices as pointing between characters



Compound data types

- Lists
  - `[1, 2.0, 'Tie my shoe']`
- Tuples
  - `(3, 4.0, 'Shut the door')`
- Dictionaries
  - `{'Clark':'Superman', 'Bruce':'Batman', 'life':42}`

Lists

```
a = ['spam', 'eggs', 100, 1234]

a[0] → 'spam'
a[3] → 1234
a[-2] → 100

a[1:-1] → ['eggs', 100]

a[:2] + ['bacon', 2*2] → ['spam', 'eggs', 'bacon', 4]

2 * a[:3] + ['Boo!'] →
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

Notes about lists

- All slice operations return a new list containing the requested elements (shallow copy)
- Lists are mutable – possible to change elements
- Simultaneous assignment → `[a, b, c] = [1, 2, 3]`
- Other ways to change lists
  - Assignment to slice → `x[0:2] = [1, 12]`
  - Remove elements → `x[0:2] = []`
  - Insert elements → `x[1:1] = ['jake', 'finn']`
  - But: perhaps better to use list functions
- Lists can be `nested`
  - [[1, 2], [3,4], [5,6]]
- `len()` also works on lists

List methods

`x = [1, 2, 3]`

`x.append(2)` → adds number 2 as fourth element

`x.count(2)` → counts the occurences of 2

`x.index(3)` → returns the index of first occurrence of 3

`x.insert(2, 5)` → inserts 5 after the 2nd index position

`x.pop()` → returns the last element and removes it

`x.reverse()` and `x.sort()` → in-place sorting

Lists can be used as stacks and queues.

Boolean

- False
  - `None`
  - `False`
  - Zero of any numeric type
  - Any empty sequence
  - Any empty mapping
- True
  - `True`
  - Everything else

Boolean Operators

- Operators (`and, or, not`)
- Comparisons (`< <= > >= == !=`)
- Object identity (`is, is not`)

The while loop

- `while <<condition>>:`
- The while loop executes as long as the condition remains true
- Body of the loop is indented, Python's way of grouping statements
  - By convention, indentation is 4 spaces (not tabs)
  - Rule: long as you're consistent...
  - However: source of frustrating errors for beginners

What do the following code snippets do?

```
a = 100

while a > 0:
   print a
   a = a - 1
```

```
a, b = 0, 1

while b < 10:
   print b
   a, b = b, a + b
```

if-elif-else

```
if <<condition>>:
   ...statements...
elif <<condition>>:
   ...statements...
else:
   ...statements...
```

```
if x < 0:
   x = 0
   print “Negative changed to zero”
elif x == 0:
   print “Zero”
elif x == 1:
   print “Single”
else:
   print “More”if
```

for

- In Python, **`for`** is an iterator for sequences
  - Acts on strings, lists, tuples
  - Therefore: different from C, Java, PHP

```
for <<var>> in <<sequence>>:
   ...statements...
```

```
fruits = ['apple', 'banana', 'orange']
for dessert in fruits:
   print “I ate %s for dessert.” % dessert
```

range()

- The **`range()`** function generates a list of an arithmetic progression of numbers
  - `range(5)` → `[0, 1, 2, 3, 4]`
  - `range(5, 10)` → `[5, 6, 7, 8, 9]`
  - `range(0, 10, 2)` → `[0, 2, 4, 6, 8]`
  - `range(10, 0, -2)` → `[10, 8, 6, 4, 2]`
- Use **`range()`** together with **`for`**

What does the following code snippet do?

```
a = ['Mary', 'had', 'a', 'little', 'lamb']

for i in range(len(a)):
   print i, a[i]
```

break and continue

Just like in C, breaks out of the smallest enclosing **`for`** or **`while`** loop

The **`continue`** statement, also borrowed from C, continues with the next iteration of the loop

pass

The **`pass`** statement does nothing.

It can be used when a statement is required syntactically but the program requires no action.

Example use: creating minimal classes.

Functions in Python

```
def <<function-name>>([<<parameters>>]):
    ...statements...
    [return <<value>>]
```

```
def fib(n):    # write Fibonacci series up to n
    """Print a Fibonacci series up to n."""
    a, b = 0, 1
    while a < n:
        print a,
        a, b = b, a+b
```

Sample functions

```
# defining functions
def origin_distance(x,y):
    return math.sqrt(x*x + y*y)

def print_var(name, value):
    print “Value of”, name, “is”, value

def func_with_no_arg():
    return 42

# calling functions
dist = origin_distance(3,4)

print_var(“x”, 42)

answer = func_with_no_arg()
```

More on this tomorrow!

A quick look at Python object-orientation

```
from math import sqrt

class Coord(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def origin_distance(self):
        return sqrt(self.x**2 + self.y**2)

    def is_at_origin(self):
        return self.origin_distance() == 0

c1 = Coord(10,20)
print c1.origin_distance()
```

Writing scripts with IDLE

Next...

- Functions in-depth
- Data structures in-depth
- Exceptions
- Libraries
- Modules
- Debugging

Questions?