



Introduction to Python

Programming Languages

Adapted from Tutorial by
Mark Hammond

Skippi-Net, Melbourne, Australia

mhammond@skippinet.com.au

<http://starship.python.net/crew/mhammond>



What Is Python?

- Created in 1990 by Guido van Rossum
 - While at CWI, Amsterdam
 - Now hosted by centre for national research initiatives, Reston, VA, USA
- Free, open source
 - And with an amazing community
- Object oriented language
 - “Everything is an object”



Why Python?

- Designed to be easy to learn and master
 - Clean, clear syntax
 - Very few keywords
- Highly portable
 - Runs almost anywhere - high end servers and workstations, down to windows CE
 - Uses machine independent byte-codes
- Extensible
 - Designed to be extensible using C/C++, allowing access to many external libraries



Python: a modern hybrid

- A language for scripting and prototyping
- Balance between extensibility and powerful built-in data structures
- **genealogy:**
 - Setl (NYU, J.Schwartz et al. 1969-1980)
 - ABC (Amsterdam, Meertens et al. 1980-)
 - Python (Van Rossum et all. 1996-)
- **Very active open-source community**




Prototyping

- Emphasis on experimental programming:
- Interactive (like LISP, ML, etc).
- Translation to bytecode (like Java)
- Dynamic typing (like LISP, SETL, APL)
- Higher-order function (LISP, ML)
- Garbage-collected, no ptrs (LISP, SNOBOL4)



Prototyping

- Emphasis on experimental programming:
- Uniform treatment of indexable structures (like SETL)
- Built-in **associative structures** (like SETL, SNOBOL4, Postscript)
- Light syntax, indentation is significant (from ABC)



Most obvious and notorious features

- Clean syntax plus high-level data types
 - Leads to fast coding
- Uses white-space to delimit blocks
 - Humans generally do, so why not the language?
 - Try it, you will end up liking it
- Variables do not need declaration
 - Although not a type-less language



A Digression on Block Structure

- There are three ways of dealing with IF structures
 - Sequences of statements with explicit end (Algol-68, Ada, COBOL)
 - Single statement (Algol-60, Pascal, C)
 - Indentation (ABC, Python)



Sequence of Statements

- IF condition THEN
 stm;
 stm;
 ..
ELSIF condition THEN
 stm;
 ..
ELSE
 stm;
 ..
END IF;
next statement;



Single Statement

- IF condition THEN
 BEGIN
 stm;
 stm;
 END
ELSE IF condition THEN
 BEGIN
 stm;
 END;
ELSE
 BEGIN
 stm;
 END;
 ..
END;
next-statement;

Indentation

- IF condition:
 stm;
 stm;
 ..
- ELSIF condition:
 stm;
 ..
- ELSE:
 stm;
 ..
- next-statement

Pythonwin

- These examples use Pythonwin
 - Only available on Windows
 - GUI toolkit using Tkinter available for most platforms
 - Standard console Python available on all platforms
- Has interactive mode for quick testing of code
- Includes debugger and Python editor



Interactive Python

- Starting Python.exe, or any of the GUI environments present an interactive mode
 - `>>>` prompt indicates start of a statement or expression
 - If incomplete, `. . .` prompt indicates second and subsequent lines
 - All expression results printed back to interactive console



Variables and Types (1 of 3)

- Variables need no declaration
- `>>> a=1`
`>>>`
- As a variable assignment is a statement, there is no printed result
- `>>> a`
`1`
- Variable name alone is an expression, so the result is printed

Variables and Types (2 of 3)

- Variables must be created before they can be used
- ```
>>> b
Traceback (innermost last):
 File "<interactive input>", line
 1, in ?
NameError: b
>>>
```
- Python uses exceptions - more detail later

## Variables and Types (3 of 3)

---

- Objects always have a type
- ```
>>> a = 1
>>> type(a)
<type 'int'>
>>> a = "Hello"
>>> type(a)
<type 'string'>
>>> type(1.0)
<type 'float'>
```


Assignment versus Equality Testing

- Assignment performed with single =
- Equality testing done with double = (==)
 - Sensible type promotions are defined
 - Identity tested with `is` operator.
- ```
>>> 1==1
1
>>> 1.0==1
1
>>> "1"==1
0
```

## Simple Data Types

---

- Strings
  - May hold any data, including embedded NULLs
  - Declared using either single, double, or triple quotes
  - ```
>>> s = "Hi there"
>>> s
'Hi there'
>>> s = "Embedded 'quote'"
>>> s
"Embedded 'quote'"
```

Simple Data Types

- Triple quotes useful for multi-line strings
- ```
>>> s = """ a long
... string with "quotes" or
anything else"""
>>> s
' a long\012string with "quotes"
or anything else'
>>> len(s)
45
```

## Simple Data Types

---

- Integer objects implemented using C longs
  - Like C, integer division returns the floor
  - ```
>>> 5/2
2
```
- Float types implemented using C doubles
 - No point in having single precision since execution overhead is large anyway

Simple Data Types

- Long Integers have unlimited size

- Limited only by available memory

- `>>> long = 1L << 64`

- `>>> long ** 5`

- `2135987035920910082395021706169552114602704522356`

- `652769947041607822219725780640550022962086936576L`

High Level Data Types

- Lists hold a sequence of items

- May hold any object
- Declared using square brackets

- `>>> l = []# An empty list`

- `>>> l.append(1)`

- `>>> l.append("Hi there")`

- `>>> len(l)`

- `2`

High Level Data Types

```
○ >>> l
    [1, 'Hi there']
>>>
>>> l = ["Hi there", 1, 2]
>>> l
    ['Hi there', 1, 2]
>>> l.sort()
>>> l
    [1, 2, 'Hi there']
```

High Level Data Types

- Tuples are similar to lists
 - Sequence of items
 - Key difference is they are immutable
 - Often used in place of simple structures
- Automatic unpacking
- >>> `point = 2,3`
>>> `x, y = point`
>>> `x`
2



High Level Data Types

- Tuples are particularly useful to return multiple values from a function
- `>>> x, y = GetPoint()`
- As Python has no concept of byref parameters, this technique is used widely



High Level Data Types

- Dictionaries hold key-value pairs
 - Often called maps or hashes. Implemented using hash-tables
 - Keys may be any immutable object, values may be any object
 - Declared using braces
- `>>> d={}`
`>>> d[0] = "Hi there"`
`>>> d["foo"] = 1`

High Level Data Types

- Dictionaries (cont.)

- ```
>>> len(d)
2
>>> d[0]
'Hi there'
>>> d = {0 : "Hi there", 1 :
"Hello"}
>>> len(d)
2
```

## Blocks

---

- Blocks are delimited by indentation

- Colon used to start a block
- Tabs or spaces may be used
- Mixing tabs and spaces works, but is discouraged

- ```
>>> if 1:
...     print "True"
...
True
>>>
```



Blocks

- Many hate this when they first see it
 - Most Python programmers come to love it
- Humans use indentation when reading code to determine block structure
 - Ever been bitten by the C code?:
- `if (1)`

```
    printf("True");  
    CallSomething();
```



Looping

- The `for` statement loops over sequences
- ```
>>> for ch in "Hello":
... print ch
...
H
e
l
l
o
>>>
```

## Looping

---

- Built-in function `range()` used to build sequences of integers
- ```
>>> for i in range(3):  
...     print i  
...  
0  
1  
2  
>>>
```

Looping

- `while` statement for more traditional loops
- ```
>>> i = 0
>>> while i < 2:
... print i
... i = i + 1
...
0
1
>>>
```



## Functions

---

- Functions are defined with the `def` statement:
- ```
>>> def foo(bar) :  
...     return bar  
>>>
```
- This defines a trivial function named `foo` that takes a single parameter `bar`

Functions

- A function definition simply places a function object in the namespace
- ```
>>> foo
<function foo at fac680>
>>>
```
- And the function object can obviously be called:
- ```
>>> foo(3)  
3  
>>>
```

Classes

- Classes are defined using the `class` statement
- ```
>>> class Foo:
... def __init__(self):
... self.member = 1
... def GetMember(self):
... return self.member
...
>>>
```

## Classes

---

- A few things are worth pointing out in the previous example:
  - The constructor has a special name `__init__`, while a destructor (not shown) uses `__del__`
  - The `self` parameter is the instance (ie, the `this` in C++). In Python, the `self` parameter is explicit (c.f. C++, where it is implicit)
  - The name `self` is not required - simply a convention

## Classes

---

- Like functions, a class statement simply adds a class object to the namespace
- ```
>>> Foo
<class __main__.Foo at 1000960>
>>>
```
- Classes are instantiated using call syntax
- ```
>>> f=Foo()
>>> f.GetMember()
1
```

## Modules

---

- Most of Python's power comes from modules
- Modules can be implemented either in Python, or in C/C++
- `import` statement makes a module available
- ```
>>> import string
>>> string.join( ["Hi", "there"] )
'Hi there'
>>>
```

Exceptions

- Python uses exceptions for errors
 - `try / except` block can handle exceptions
- ```
>>> try:
... 1/0
... except ZeroDivisionError:
... print "Eeek"
...
Eeek
>>>
```

## Exceptions

---

- `try / finally` block can guarantee execute of code even in the face of exceptions
- ```
>>> try:  
...     1/0  
...     finally:  
...         print "Doing this anyway"  
...  
Doing this anyway  
Traceback (innermost last):  File  
"<interactive input>", line 2, in ?  
ZeroDivisionError: integer division or modulo  
>>>
```

Threads

- Number of ways to implement threads
- Highest level interface modelled after Java
- ```
>>> class DemoThread(threading.Thread):
... def run(self):
... for i in range(3):
... time.sleep(3)
... print i
...
>>> t = DemoThread()
>>> t.start()
>>> t.join()
0
1 <etc>
```

## Standard Library

---

- Python comes standard with a set of modules, known as the “standard library”
- Incredibly rich and diverse functionality available from the standard library
  - All common internet protocols, sockets, CGI, OS services, GUI services (via Tcl/Tk), database, Berkeley style databases, calendar, Python parser, file globbing/searching, debugger, profiler, threading and synchronisation, persistency, etc



## External library

---

- Many modules are available externally covering almost every piece of functionality you could ever desire
  - Imaging, numerical analysis, OS specific functionality, SQL databases, Fortran interfaces, XML, Corba, COM, Win32 API, etc
- Way too many to give the list any justice



## Python Programs

---

- Python programs and modules are written as text files with traditionally a `.py` extension
- Each Python module has its own discrete namespace
- Name space within a Python module is a global one.



## Python Programs

---

- Python modules and programs are differentiated only by the way they are called
  - .py files executed directly are programs (often referred to as scripts)
  - .py files referenced via the `import` statement are modules



## Python Programs

---

- Thus, the same .py file can be a program/script, or a module
- This feature is often used to provide regression tests for modules
  - When module is executed as a program, the regression test is executed
  - When module is imported, test functionality is not executed



## More Information on Python

---

- Can't do Python justice in this short time frame
  - But hopefully have given you a taste of the language
- Comes with extensive documentation, including tutorials and library reference
  - Also a number of Python books available
- Visit `www.python.org` for more details
  - Can find python tutorial and reference manual



## Scripting Languages

---

- What are they?
  - Beats me 😊
  - Apparently they are programming languages used for building the equivalent of shell scripts, i.e. doing the sort of things that shell scripts have traditionally been used for.
  - But any language can be used this way
  - So it is a matter of convenience





## Characteristics of Scripting Languages

---

- Typically interpretive
  - But that's an implementation detail
- Typically have high level data structures
  - But rich libraries can substitute for this
  - For example, look at GNAT.Spibol
- Powerful flexible string handling
- Typically have rich libraries
  - But any language can meet this requirement



## Is Python A Scripting Language?

---

- Usually thought of as one
- But this is mainly a marketing issue
  - People think of scripting languages as being easy to learn, and useful.
- But Python is a well worked out coherent dynamic programming language
  - And there is no reason not to use it for a wide range of applications.



## An Example in Python

---

### Scramble Sort



### Scramble Sort

---

- The scramble sort problem deals with a list of mixed integers and strings.
- The integers are to be sorted in order
- The strings are to be sorted in order
- With the constraint that integers appear where integers were in the original list, and strings appear where strings appeared in the original list.

## Setting Up The Data

---

- `>>> list = [1,10,'abc','hello',3, 'car', 0, 'aardvark']`
- `>>> list`
- `[1, 10, 'abc', 'hello', 3, 'car', 0, 'aardvark']`
- `>>> len (list)`
- `8`

## Defining The Sort Function

---

- `>>> def sort(l):`
- `... for j in range(0,len(l)):`
- `... for k in range(j+1,len(l)):`
- `... if (type(l[j])==type(l[k])) and`  
`(l[j]>l[k]):`
- `... t=l[k]`
- `... l[k]=l[j]`
- `... l[j]=t`
- `... return sort(l)`
- `... return l`
- `...`



## Running the function

---

- `>>> sort (list)`
- `[0, 1, 'aardvark', 'abc', 3, 'car', 10, 'hello']`
- `>>>`



## Another Problem, Digital Roots

---

- Given a (possibly very long) decimal number
- Sum up all the digits
- Repeat the process until the result is less than 10
- This result is the digital root



## Observation

---

- This is equivalent to casting out 9's
- The result is the number mod 9, except that we get 9 instead of 0 for non-zero input.
- Easy in Python because we can handle large numbers directly



## Set Up The Data

---

- `>>> num = 123 ** 123`
- `>>> num`
- 114374367934617190099880295228066  
276746218078451850229775887975052  
369504785666446606568365201542169  
649974727730628842345343196581134  
895919942820874449837299476648958  
359023796078549041949007807220625  
356526926729664064846685758382803  
100766740220839267L
- `>>>`



## Define The Function

---

- `>>> def digital(n):`
- `... if n==0:`
- `... return 0;`
- `... if n%9==0:`
- `... return 9;`
- `... return n%9;`
- `...`



## Some Examples of Digital Roots

---

- `>>> digital(0)`
- `0`
- `>>> digital(18)`
- `9`
- `>>> digital (num)`
- `9`
- `>>> num=num+7*9999-3`
- `>>> digital(num)`
- `6L`



## Note on Input-Output

---

- For simplicity, I have omitted input output details here
- But when you do the problem, you should indeed handle the input and output formatting as specified in the problem
- That's only fair in comparing Python with other languages