What Is Python?

- Created in 1990 by Guido van Rossum

- About the origin of Python, Van Rossum wrote in 1996:

-     Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office … would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

-

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.

Design Philosophy

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

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

Most obvious and notorious features

- Clean syntax plus high-level data types
  - Leads to fast coding (First language in many universities abroad!)
- 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

Productivity!

- Reduced development time
  - code is 2-10x shorter than C, C++, Java

- Improved program maintenance
  - code is extremely readable

- Less training
  - language is very easy to learn

What is it used for?

- rapid prototyping
- web scripting
- throw-away, ad hoc programming
- steering scientific applications
- extension language
- XML processing
- database applications
- GUI applications
- A Glue Language

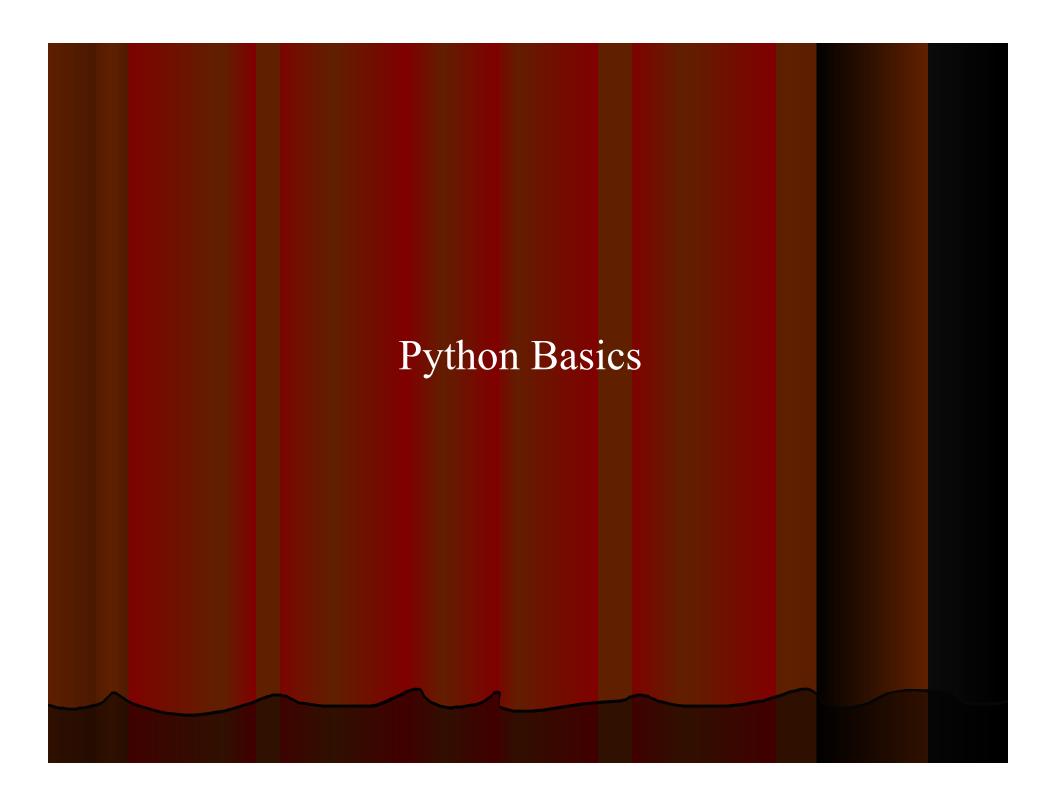# PYTHON COMPARED TO OTHER LANGUAGES

Python vs. Perl

- Easier to learn
  - important for occasional users
- More readable code
  - improved code maintenance
- Fewer "magical" side effects
- More "safety" guarantees
- Better Java integration
- Less Unix bias

Python vs. Tcl

- Real datatypes, object-orientation
- More differentiated syntax
- Much faster (even than Tcl 8.x)
- Less need for C extensions
- C extensions don't redefine syntax
  - hence fewer extension conflicts
- Better Java integration
- Python uses Tk as de-facto GUI std

Python vs. Java

- Code 5-10 times more concise
- Dynamic typing
- Much quicker development
  - no compilation phase
  - less typing
- Yes, it runs slower
  - but development is so much faster!
- Use Python with Java: Jython!

Python Basics

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

Variables and Types

- 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

- 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

- 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`
    `2135987035920910082395021706169552114602704522356652769947041607822219725780640550022962086936576L`

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
  - Maxing 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, comedilib, serial, parallel, opengl, opencv, wxpython, gtk, qt, tkinter etc
- Way too many to give the list any justice