Python for Fundamentals

Bob Hancock

hancock.robert@gmail.com
bobhancock.org

Section I

The Python Language

What Is Python

- A multi-purpose interpreted language invented by Guido von Rossum in 1990.
- You can run the interpreter from the command line.
- Rapid iteration.
- Extensive library support.
- A cooperative community.
- We will use Python 2.7x as a reference.

Python 2 or 3?

- Python 2.7.6 is the last of the 2 series.
- Python 3 is where the new language development occurs
- 2 and 3 can be incompatible because they handle Unicode differently, among other things.
- There was a conscious decision to make 3 incompatible to fix legacy problems.
- I still use both versions and it depends upon the library.
- Get to know Python3 if you can.

Get Python

- Go to http://python.org/downloads/
- For Linux, download the source code
- Perform the normal GNU build sequence.
- --prefix=/usr/local/python{version}

Start Python

- At the command prompt type “python”
- For help type:
  - import sys
  - help(sys.maxsize)
- You need to import the module before you can issue a help request.
- type “import this”
  - see what happens

Hello, world!

```
> python
>>> print("Hello, world!")
Hello, world!
>>> print("Hello,"+" world!")
Hello, world!
>>> print("{h}, {w}!".format(h="Hello", w="world"))
Hello, world!
```

Part 1.1

Basic Syntax

Basic Syntax

- It has a C like syntax
- You must use consistent indentation.
- No semicolons for line ends.
- Readability counts.

```
>>> a = "bob"\
... +" hancock"
>>> a
“bob hancock”
```

Numbers

```
>>> 7/3 #Integer division returns fractions
2.3333333333333335
>>> 7//3 #You can get the floor
2
>>> 7.0//3 #Promotes to the more complex type
2.0
```

Assignment

- x =1
- x = y = z = 0
- x, y = 1, 2
- a = x

Assignment

- Variables must be defined before use.

```
>>> a

>>>

>>> a

>>> a="Python"

>>> a

'Python'
```

Part 1.2

Data Types

None

- Denotes a null object.
- This is returned by functions that do no explicitly return a value.
- Frequently used as the default value for optional arguments.
- Has no attributes.
- Evaluates to False in boolean expressions.

Numeric

- boolean
- integer (Python 2.x only)
- long integers
- floating-point numbers
- complex numbers

boolean

- Represented by True or False.
- Numerical values 1 is True and 0 is False.

```
x = 1

if x:
    print("x is True")
else:
    print("x is False")
```

datatypes/boolean.py

Integers

- In Python 3 all integers are in effect long.
- `sys.int_info(bits_per_digit=30, sizeof_digit=4)`
- Python integers are stored internally in base 2**int_info.bits_per_digit.
- sys.maxsize – 2**63 – 1 on a 64-bit system.
- sizeof_digit is the size in bytes of the C type used to represent a digit.

Floating-point

- IEEE 754 which provides approximately 17 digits of precision.
- Doesn't support 32-bit single precision.
- If you need precise precision or control, you can use numpy.
- See Matopolis for a comparison with Matlab.

Complex (optional)

- Complex numbers are always represented as two floating point numbers, the real and imaginary part.
- To extract these parts from a complex number z, use z.real and z.imag.

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Sequence Types

- Ordered sets of objects indexed by non-negative numbers.
- Lists
- Tuples
- Strings

Sequence Types

- All sequences support iteration.
- strings and tuples are immutable.
- strings are sequences of characters.
- tuples are sequences of arbitrary objects.
- lists are mutable.
- lists are sequences of arbitrary objects.

List

- An iterable sequence.

```
L0 = [1, 2, 3, 4]
L1 = [[1,2,3], [4], [5,6]]
L3 = [“palin”, (1,2), [1], {“monty”:1}]
```

- The elements can be accessed by zero based indexes.
- `print(L3[1][1])`

  … 2

List

- `list(s)` converts s to a list.
- `s.append(x)` appends element to the end.
- `s.extend(t)` appends new list t to the end.
- What if you append a list to a list instead of using extend?
- `s.clear()`r clears all members of the list.
- `s.pop()` stack like behavior.
- `s.insert(i, x)` insert x at index I.

Strings

- “Dead” #double quotes
- 'Parrot” #single quotes
- '”Don't, quote me”'
- 'Don\'t quote me'

  a = “This is a long string\n\
  that spans several lines, but\n\
  prints with no \
  problem?”
- The white space from the left is significant.

Strings

- Starting with Python 3.0 all strings support Unicode.
- You can embed Unicode runes using Python-Unicode escaping.

```
>>> 'Hello\u0020World !'
'Hello World !'
```

- Or the actual Unicode character.

```
>>> print("Hello, 世界")
Hello, 世界
```

Strings

- You can convert a string to a string of bytes with encode().
- >>> "Äpfel".encode('utf-8')
  b'\xc3\x84pfel

Triple Quotes

- Ends of lines do not need to be escaped when using triple-quotes, but they will be included in the string.

```
print(“””\This is a
long string where
        white space is significant”)
```

```
This is a
long string where
      whitsepace is significant.
```

- Used in procedure doc strings.

String Concatenation

```
>>> print("spam" " and" " eggs")
spam and eggs


>>> a = "Bicycle" + " Repairman"
>>> print(a)
Bicycle Repairman

>>> print("  Pyt   ".strip() + "hon")
Python
```

String Subscripting

```
>>> a = "spam baked beans and spam"
>>> a[5]
'b'
>>> a[5:10]
'baked'
>>> a[-1]
'm'
>>> a[5:11]+a[21:]
'baked spam'
```

Strings are Immutable

```
>>> a[3] = "z"

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
assignment
```

String Methods

- help(str) NOT help(string) is required for the list of methods.
- There are 38 methods on string.
- s.find(sub [, start, end])
- s.split([sep [,maxsplit]])
- s.zfill(width) pads with zeros on the left.
- s.join() joins strings in t with separator s.
  - This is faster than concatenation for large strings.

slicing

```
>>> L = [1,2,3,4,5,6]
>>> L[2:4]
[3, 4]
>>> L[:4]
[1, 2, 3, 4]
>>> L[-1]
6
```

Part 1.3

Mapping Types

Mapping Types

- A mapping object is an arbitrary collection of objects that are indexed by another nearly arbitrary collection of objects.
- Unlike a sequence, mapping objects are unordered.
- They can be mapped by any immutable object type.
- Mapping objects are mutable.
- Dictionaries are the only built-in mapping type.
- With lists and strings these are the most frequently used data storage elements.

Dictionaries

- Python's version of a hash map or associative array.
- The retrieval time of a dictionary in O(1).
- Think of it as an unordered set of keys and values.
  - Keys must be unique.
- `list(d.keys())` for a list of all keys.
- `sorted(d.keys())` for a list of sorted keys.
- `del d[key]` to delete an entry.
- `key in d` returns True or False (membership test).

Set Types

- A set is an unordered collection of unique items.
- No indexing or slicing operations.
- s.difference(t) returns all the items in s not in t.
- s.intersection(t) returns all items ins both s and t.
- s.isdisjoint(t) returns True if s and t have not items in common.
- s.issubset(t) returns True if s is a subset of t.
- s.untion(t) returns all items in s and t.

Copying Objects

```
l = [ 1,2,3, [4,5,6] ]
l_copy = l
l_copy.append("end")
print(l)
   [ 1,2,3, [4,5,6], "end" ]
```

New or DeepCopy

- This create a new copy of the original list.

  l = [1,2,3, [4,5,6]]

  l_new = lst(l)

- You can also make a deep copy.

  import copy
  l_deep = copy.deepcopy(l)

Part 1.4

Flow Control

if

```
x = 42

if x < 0:
    x = 0
    print('Negative changed to zero')
```

for

```
w = words[ “dead”, “parrot”, “spam” ]

for w in words:
    print(w)

…

dead

parrot

spam
```

for

```
w = words[ “dead”, “parrot”, “spam” ]

for w, i in enumerate(words):
    print(i, w)

…

0 dead

1 parrot

2 spam
```

range

- Generates automatic arithmetic progressions.

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

range returns an iterable

```
>>> print(range(10))
range(0, 10)
```

- Range returns an iterable object.
- You a for statement to access results, or
- A functionsthat takes an iterable argument.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

while

```
count = 0
while count < 5:
      count += 1
      print("Incremented count to
         "+str(count))
else:
      print("Condition is false")
```

- else is executed when the while condition becomes false.
- I've never used the else clause of while.

while

```
while True:
      read from a socket
      if some condition:
        break
```

- Endless loop.
- You assume that something will always be true and you only want to break out if something extraordinary happens.

break

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n,'equals',x, '*', n//x)
            break
    else:
        print(n, 'is a prime number')
```

flow/break.py

- N.B. the else statement is part of the for loop.

pass

```
while True:
   pass  # Busy-wait for keyboard
         # interrupt (Ctrl+C)
```

flow/pass.py

- used when a statement is required syntactically but the program requires no action
- Useful when developing a place holder function.

Part 1.5

Functions:
The Building Blocks

Functions

```
def sqrt(x):
    return x*x
```

- To invoke the function, use the name and place the arguments in parantheses.

```
s = sqrt(2)
print(s)
...  4
```

Returning Multiple Values

```
def foo(x):
    a = x*2
    b = x*3
    return (a,b)
```

- The return value is a tuple that can be unpacked.

```
x, y = foo(2)
print(x, y)
   4  6
```

Default Arguments

```
def bar(x, y=4):
    a = x*2
    b = y*3
    return (a,b)
```

- Default values can be omitted when calling

```
x, y = bar(2)
print(x, y)
   4  12
x, y = bar(2,6)
print(x, y)
  4   18
```

Function Scope

- Variables create within a function are local.
- To access a global variable use the global keyword.

```
a = 100
def foo():
    global a
    a = 99
    print(a)

print(a)
    100
foo()
   99  ← result of print inside of foo
print(a)
    99
```

Pass by Value or Reference

- Pass by value means that a copy of the arguments are supplied to the function.
- Changes within the function will not affect the variables used in calling the function.
- Pass by reference means that a pointer to the arguments are supplied to the function.
- The function can mutate the data to which the references point.
- What does Python do?

Mutable Arguments

```
>>> def mut(l):
...     l.append("end")
>>> lst = [1,2,3]
>>> print(lst)
[1, 2, 3]
>>> mut(lst)
>>> print(lst)
[1, 2, 3, 'end']
```

Arguments

- If the function argument is immutable, it will not change.
- If the function argument is mutable, you can change the calling variable.
- If you pass a mutable argument, make a copy within the function and return the modified object.

Part 1.6

Exception Handling

Exceptions

- Exceptions indicate errors and break out of the normal control flow of a program.
- An exception is raised with
  raise Exception([value])
- For example:

```
raise RuntimeError("Unrecoverable Error")
```

try..except

```
try:
    do something
except IOError as e:
    statements
except NameError a e:
    statements
else:
    code in try block did not raise exception, so
    statements
finally:
    statements
    always executed whether exception or not
```

try..except

```
>>> try:
...     f = open("/home/rhancock/testfile")
... except IOError as e:
...     print(e)
... else:
...     print("else")
... finally:
...     print("finally done")
```

Section II

Advanced Topics

Part 2.1

Generator Functions

Generators

- A generator is a function that produces a sequence of results for iteration. **gen_co/generators_1.py**

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

m = 10
x = countdown(m)
print(x)
for i in range(3):
    print("Countdown: {d}".format(d=x.next()))
```

Generators

- Function returns a generator object.
- The generator object executes the function when next is called.
- The function executes until it reaches yield statement.
- Yield produces a result and execution halts until the next invocation of next.
- Let's step through the code in debug.

itertools.chain()

```
def chain(*iterables):
    # chain('ABC', 'DEF') --> A B C D E F
    for it in iterables:
        for element in it:
            yield element
```

Generator Pipelines

- Like shell pipes in UNIX.
- Create a function that receives a value and emits a value.
- You can hook these together and insert other generators as you alter your code.
- How does this affect Bigfile processing?
- Let's look at the timings.
- **xferlog/bigfile_pipeline_1.py**

Yield as an Expression

- Python 2.5 (PEP-342) added yield as an expression.

```
def grep(pattern):
    while True:
        line = (yield)
        if pattern in line:
            print(line)
```

- What is the difference between a statement and an expression?
- So what does this do for me?

Part 2.2

Coroutines

Coroutines

- Using yield as an expression allows you to create coroutines.
- Coroutines execute when values are sent to them.
- (yield) returns the value.
- Generators only produce values.
- Coroutines consume values and return.
- Generators are about iteration.
- Coroutines are about consuming values.

Coroutines

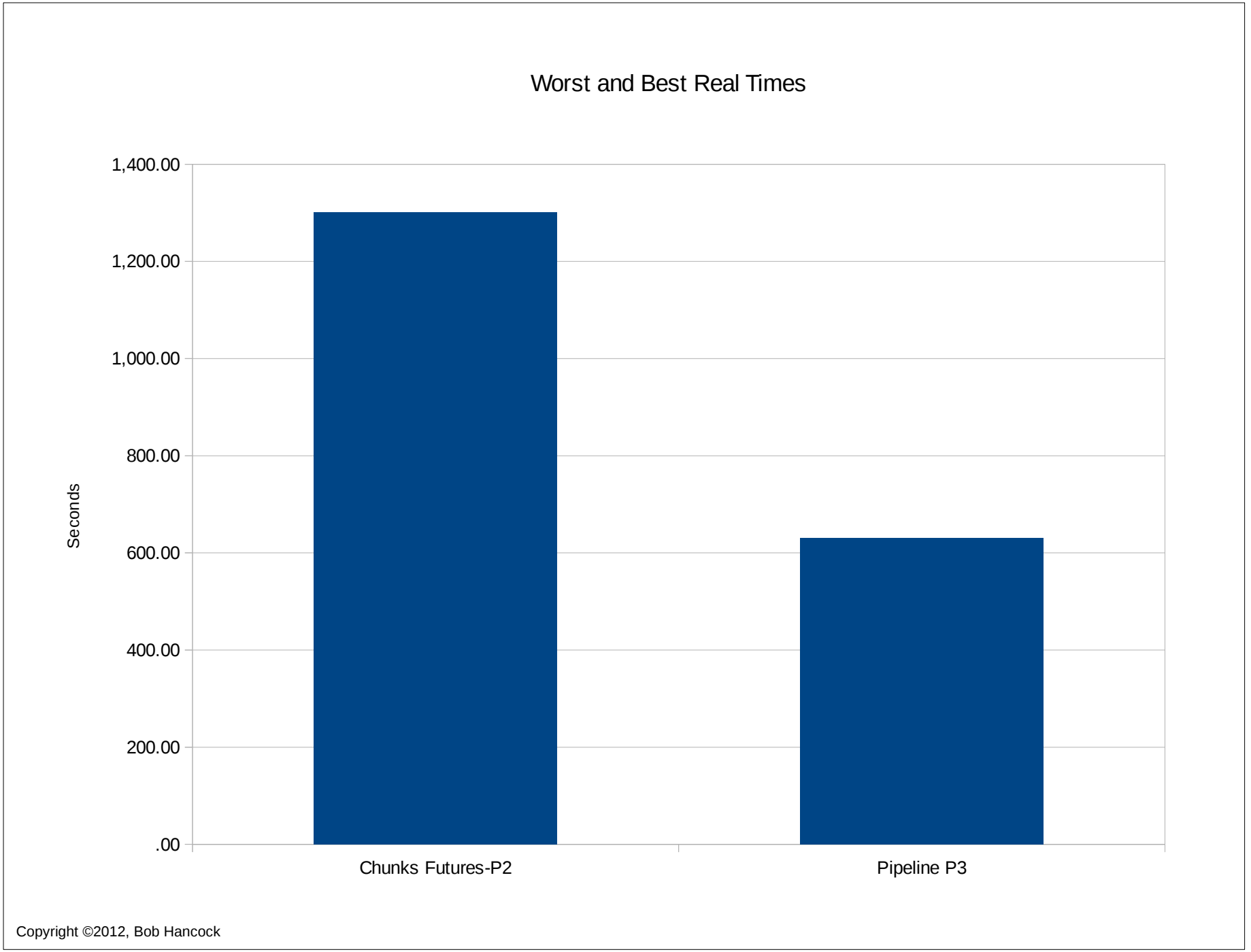
- What is a coroutine in computer science?
- The concept of coroutines has been around since 1958 – M. E. Conway
- Coroutines are functions that save control state between calls.
- Why did threads overtake coroutines?
- Prime the coroutine with a decorator (Thanks to Dave Beazley).
- **gen_co/coroutines_1.py**

Coroutines

- What would this do for the bigfile processing?
- **xferlog/bigfile_pipeline_2.py**
- Let's look at the numbers.

Coroutines

- Python coroutines can be used to implement concurrency.
- An event loop can send data to a collection of coroutines that carry out diverse tasks.
- Python generators would be great if we had non-blocking IO
- For that we can use gevent.



Part 2.3

Modules

Creating Your Own Modules

- Once you have a group of functions that you want to re-use, package your own modules.
- What main() does.
- Modules must be contained in a directory.
- The presence of __init__.py in the directory tells the interpreter that this can be imported.
- Be sure to set your PYTHONPATH.
- Be careful to not use reserved words or module names.
- packages directory.

Part 2.4

Classes

What is a class?

- Classes bind data with code.
- The bound functions are called methods when they are part of a class.
- Class is a noun.
- Methods are verbs.

pypackages/dry/system.py

Part 2.5

os module

Interface to os Calls

- help() is not sufficient for this module.
- http://docs.python.org/3/

Section III

Text and Files

Split String on Delimiters

- What does this print?

```
line = "abc,def,ghi,jkl"
s = line.split(",")
print(s)
```

text/split_single_delimiter.py

Split String on Multiple Delimiters

- What does this print?

```
import re


line = "abc def ghi; jkl mno,pqr,    foo"
s = re.split(r'[;,\s]\s*', line)
print(s)
```

Match Text at Start or End

```
filename = "foo.txt"
print(filename.endswith("txt")) # True
print(filename.startswith("foo")) # True
print(filename.startswith("bar")) # False
print(filename.startswith(("bar", "bob", "foo") )) # True
print(filename.startswith("o", 1)) # True
print(filename.startswith("oo", 1, 3)) # True
```

Match with Shell Wildcards

```
from fnmatch import fnmatch, fnmatchcase
fnmatch("foo.txt", "*.txt") # True
fnmatch("foo.txt", "?oo.txt") # True
fnmatch("ops32.csv", "ops[0-9]*") # True
fnmatchcase("foo.txt", "*.TXT") # False
```

Find Substring

line = "abc def ghi; jkl mno,pqr,   foo"

- line.find("ghi") # 8
- line.find(",") # 20 finds first occurence
- line.find("X") # -1 not found
- line.find("ef", 3, -1) # 5 first occurence in range but index of full string

Stripping Characters

```
line = "Xabc def ghi; jkl Xmno,pqr,    fooXX"
line.strip(";") # removes all Xs at start and end only

line = "    Hello, world!    \n"
line.strip() # 'Hello, world!'
line.rstrip() # '    Hello, world!'
line.lstrip() # 'Hello, world!    \n'
```

Interpolating Variables in a String

- string.format()
- Prefer format over C style printf() formatting.

  s = “%s %s” % “bob”, “hancock” # C style

  versus

  s = “{fn} {ln}”.format(fn=”bob”, “hancock”)
  fn=”bob”,

  s = “{fn} {ln}”.format(“hancock”, fn=”bob”)

- Why is format() better?

What is Next?

- Profiling
- Debugging
- Generators
- Coroutines
- How to decompose problems into code?
- Optimizations (Advanced)
- Case Studies
  - Multi-processor rsysnc
  - Processing large log files

Section VII

Questions