title: Introduction to Python
data-tranition-duration:
1500
css: python.css

MQT Basic Developer Training

Python

Authors:
Eric Ortega eric.j.ortega@gmail.com
Chris Plummer christopher.plummer.4@us.af.mil

Version:
v0.1 MAR 23 2016

Objectives

- Understand what Python is
- Explain common uses of Python
- Know how to user Python documentation
- Articulate some differences between 2.x and 3.x
- Run Python from the interpreter
- Run Python from a file

Why use Python

- High-level
- Interpreted
- Simple Syntax
- Cross Platform
- Batteries included

> **Note**
>
> High-level: abstraction from details of the computer, typically easier to use Interpreted: directly executes instructions Readable: easier to read, learn, and understand, which makes is more writeable. Cross Platform: runs everywhere (almost) Batteries included: boatload of software in standard library
>
> Easy to learn yet is a pretty deep language as you will see later

---

Typical uses of Python

- Prototyping
- Automation and Testing
- Vulnerability Research/Fuzzing
- Create/replicate pieces of software that you do not otherwise have access to

---

Python Versions

- Python 2.x
- Python 3.x
- They are similar but **NOT** compatable!
- We will be focusing on Python 2.7

> **Note**
>
> 2.7: last version of Python 2. 2to3, automated Python 2 to 3 code translation

---

Differences in Python 3: Print

Python 2 allows calls to print WITHOUT or with parenthesis: Print is treated as a statement rather than a function.

```
dev@laptop-c:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
(Omitted extra lines)
>>> mystr = "Hello World"
>>> print mystr
Hello World
>>>
```

In Python 3, print now a function and requires parenthesis:

```
dev@laptop-c:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
(Omitted extra lines)
>>> print mystrt
  File "<stdin>", line 1
    print mystrt
                ^
SyntaxError: Missing parentheses in call to 'print'

>>> print(mystr)
Hello World
>>>
```

Differences in Python 3: Integer Division

Python 2 treats numbers without a "." as integers Division is treated as integer division and the remainder is truncated

Setting one number as a real number will yield the correct results

```
dev@laptop-c:~$ python
>>> 3/2
1
>>> 3.0/2
1.5
```

In Python 3, division evaluates as a float by default

```
dev@laptop-c:~$ python3
>>> 3/2
1.5
```

Python Documentation

- Mostly concerned with standard library
    - https://docs.python.org/2/library/index.html
    - https://docs.python.org/3/library/index.html
- This is how you use the docs....

Python and Whitespace

Python is whitespace sensitive

- Instead of using { } like in C, code is indented
- PEP8 specifies that 4 character indent is "Pythonic"

The eternal battle: tabs or spaces?

- Do **NOT** mix spaces (/s) and tabs (/t) for indents
- Using editors with different tab settings might break the code
- Solution: Use tabs and set your IDE's tabstop to be your preferred length

---

Structure of .py files

The below script calculates the sum of a list of numbers 1-4

Pay attention to the comments showing the changes in indentation levels

```
# no indent
import math

def sumList(numbers):
    # 1 level of indent
    retVal = 0
    for num in numbers:
        # 2 level of indent
```

```
        retVal += num

    # back to 1 level
    return retVal

# back to no indent
numList = [1,2,3,4]
sumOfList = sumList(numList)
print "The sum of the elements in the list " \
    + str(numList) + " is %d" %(sumOfList)
```

Running Python: Interpreter

- Run python from command line
  - executing 'python' will run the default version
  - You may specify the version with 'python2' or 'python3' respectively
- On most linux systems, python 2 is the default

```
dev@laptop-c:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>>

dev@laptop-c:~$ python2
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>>

dev@laptop-c:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>>
```

REPL

The Python interpreter uses a REP loop for programming

- READ the user input
  - Some constructs like loops might be multiple lines
- EVALuate the input
  - attempt to perform the instruction entered
- PRINT to the screen
  - print any requested info **OR** an error with stack trace
  - print the next prompt

```
>>> mystr = "Hello World"
>>> print mystr
Hello World
>>>
```

LAB: Interpreter

Using the Interpreter

```
dev@laptop-c:~$ python
>>> mystr = 'Hello World!'
>>> print mystr
Hello World!
```

LAB: py files

Using the interpreter is great for quick snippets but who wants to type everything out each time?

Solution: Put everything in a text file named 'lab1.py'

```
mystr = 'Hello World!'
print mystr
```

And run 'lab1.py' using the python executable

```
dev@laptop-c:~$ python lab1.py
Hello World!
```

Standard Data Types

System Message: ERROR/3 (`combined.rst`, line 292)

Document or section may not begin with a transition.

Variables

- Everything is an object
- Data type is dynamic based upon current stored value
- Multiple assignment is allowed
    - a = b = c = 100
        - c = 200 will change it for both a & b also
    - a,b,c = 100,"hello", {}
        - assigns 100 to a
        - assigns "hello" to b
        - and an empty dict to c

Note

objects have a type: e.g. boolen or integer that determines what can be done with the data type(<var>) to view type of variable a-Z,A-Z, 0-9, _

Numbers

- IMMUTABLE
- Understand Decimal, Binary, Hexadecimal, Octal
    - no prefix for decimal
    - prefix with 0b, 0B for binary (e.g. 0b10 == 2)
    - prefix with 0x, 0X for hex (e.g. 0xF == 15)
    - prefix with 0, 0o, 0O for octal (e.g. 0100 == 64)
- 4 types of Numbers
    - Integers (1, 203, -3)
    - Long (decimal, hex and octal) (12345L, 0x79CFAL, 0101L)
        - suffix of L
    - Float (0.0, 17.5+e18)
    - Complex (1+3j)

Note

immutable: can not be changed, needs to be reassigned

Operators

Operator	Description	Example	Result
+	addition	5 + 8	13
-	subtraction	90 - 10	80
*	multiplication	4 * 7	28
/	floating point division	7 / 2	3.5
//	integer (truncating) division	7 // 2	3
%	modulus (remainder)	7 % 3	1
**	exponentiation	3 ** 4	81

Note

introducing python 2 Numbers

Type Conversions

- int(x, base)
- long(x, base)
- float(x)
- complex(real, imag)
- chr(x)
- unichr(x)
- ord(x)
- hex(x)
- oct(x)

LAB: Working with numbers

System Message: ERROR/3 (`combined.rst`, line 364)

Document or section may not begin with a transition.

Sequence Objects

- Sequence is a container of items accessed via an index
- Includes: Strings, lists, and tuples
- Each type supports built in functions and slicing
- Slicing
    - var[4] = 4th element
    - var[4:9] = subset consisting of elements 4-9
    - var[-4] = var[len(var) - 4] = 4th to last element
- Built in methods

- see chart: https://docs.python.org/2/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange

---

Sequence Objects: Strings

- IMMUTABLE
- Two independent types: ASCII (traditional str) and UNICODE
- String literals
- “byte” or “data” strings
- Prefixes
  - u’…’ for Unicode
  - r’…’ for raw string literal (does not interpret backslash as escape)
- Useful methods
  - upper() and lower()
  - len()
  - join() and split()
- Type conversion
  - str(x)

> **Note**
>
> strings is a sequence of characters and are immutable. Characters can be accessed one at a time with the bracket operator

---

Strings cont'd

- my_string = 'Hello World!'

```
print my_string          >> Hello World!
print my_string[0]       >> H
print my_string[0:5]     >> Hello
print my_string[6:]      >> World!
print my_string[-6:]     >> World!
print my_string[::2]     >> ????
print my_string * 2      >> ????
print my_string + my_string >> ????
```

> **Note**
>
> my_string[::2] = HloWrd my_string * 2 = Hello World!Hello World!

Strings cont'd

- my_string = 'Hello World!'

```
>> a = my_string.split(' ')
>> ['Hello', 'World!']  #a is a list split on ' '

>> ' '.join(a)          #joins the list seperated by a ' '
>> 'Hello World!'
```

> **Note**
>
> question to students: what is the value of a after ' '.join(a) answer: ['Hello', 'World!'], reason: IMMUTABLE

Strings: Changing a character

- my_string = 'Hello World!'

```
>> my_string[1] >> 'e'
>> my_string[1] = 'E'
```

- What happens? why?
- try <var>.replace('dst','src') instead

> **Note**
>
> IMMUTABLE, can't change a character

More String Commands

startswith, endswith, find, rfind, count, isalnum, strip, capatalize, title, upper, lower, swapcase, center, ljust, rjust

---

Lab: Working with strings

> **System Message: ERROR/3 (`combined.rst`, line 474)**
>
> Document or section may not begin with a transition.

Sequence Objects: Lists

- MUTABLE
- Nestable (e.g 2D Array)
- Variable Length
- Not ordered

Lists can contain elements of different types

> **Note**
>
> A list is a mutable sequence of items. Items of a list may be different types

---

Lists cont'd

- Indexing
  - how?, nested arrays?
  - index()
  - 'in'
- Updating
  - append()
  - offset (e.g. list[i])
  - insert()
  - sort(), sorted()
- Combining
  - extend() or +=
- Removing
  - remove(), del, pop()

Lab: Working with lists

> **System Message: ERROR/3 (`combined.rst`, line 512)**
>
> Document or section may not begin with a transition.

Sequence Objects: Tuples

- IMMUTABLE
- Fixed length
- Bytearray
- Common Operations

A tuple is just like a list except IMMUTABLE

> **Note**
>
> A tuple is an immutable sequence of items. Items of a tuple may be different types

---

Tuples: Things you can do

- It's a sequence object
    - splice, index

Lab: Working with tuples

> **System Message: ERROR/3 (`combined.rst`, line 540)**
>
> Document or section may not begin with a transition.

Collection Objects:

What are collection objects. Unordered collection of distinct hashable objects that may or may not have an associated value.

Dictionaries

Unordered collection of distinct IMMUTALE objects with an associated value.

- MUTABLE
- Sequence of key-value mappings
- Common Operations
- Characteristics (not ordered, values supported)

Dictionaries cont'd

```
>> dict_x = {}  #create empty dictionary
>> dict_x['one'] = 1  #add item to the dictionary
>> dict_x = {'one' : 1}  #create a dictionary with an item
```

Common Dictionary Operations

```
>> d[i] = y             #value of i is replaced by y
>> d.clear()            #removes all items
>> d.copy()             #creates a shallow copy of dict_x
>> d.fromkeys(S[,v])    #new dict from key, values
>> d.get(k[,v])         #returns dict_x[k] or v
>> d.items()            #list of (key,value) pairs
>> d.iteritems()        #iterator over (key,value) items
>> d.iterkeys()         #iterator over keys of d
>> d.itervalues()       #iterator over values of d
>> d.pop(k[,v])         #remove/return specified (key,value)
>> d.popitem()          #remove/return arbitrary (key,value)
>> d.update(E, **F)     #update d with (key,values) from E
```

Lab: Working with Dictionaries

System Message: ERROR/3 (combined.rst, line 598)

Document or section may not begin with a transition.

Sets

- MUTABLE
- Sequence of unique keys
- Common Operations
- Characteristics (not ordered, values supported)

Sets

```
>> new_set = set() #create an empty set
>> new_set = {0, 1, 2, 3, 4}
```

Common Set Operations

```
>> s.issubset(t)     #test if elements in s are in t
>> s.issuperset(t)   #test if elements in t are in s
>> s.union(t)        #new set with elements of t and s
>> s.intersection(t) #new set with common elements
>> s.difference(t)   #new set with elements in s but not t
>> s.symmetric_difference  #xor
>> s.copy()          #new shallow copy of s
>> s.update(t)       #return set s with elements from t
>> s.interesection_updtae(t)
>> s.difference_update(t)
>> s.symmetric_difference_update(t)
>> s.add(x)          #add x to set s
>> s.remove(x)       #remove x from set s
>> s.discard(x)      #remove x from set s if present
>> s.pop()           #remove arbitrary item
>> s.clear()         #remove all elements from set s
```

Lab: Working with Sets

System Message: ERROR/3 (combined.rst, line 647)

Document or section may not begin with a transition.

Conversion Functions

There are functions to convert values to datatypes if possible - float(), chr(), list(), dict(), set(), etc...

Python flow control

System Message: ERROR/3 (combined.rst, line 660)

Document or section may not begin with a transition.

Expressions

- Math Operators
- Bitwise Operators
- Comparison Operators
- Assignment Operators
- Logical Operators
- Others…(in, del, with)

Bitwise Operators

```
x << y   #returns x left-shifted y places
x >> y   #returns x right-shifted y places
x & y    #bitwise and
x | y    #bitwise or
x ^ y    #bitwise xor
~x       #returns the ones complement
```

Comparison Operators

```
x == y   #if x equals y, returns True
x != y   #if x is not y, returns True
x <> y   #same as != (deprecated)
x > y    #if x greater than y, return True
x < y    #if x less than y, return True
x >= y   #if x greater or equal than Y, return True
x <= y   #if x less or equal than Y, return True
```

LAB: Using Python Expressions

System Message: ERROR/3 (`combined.rst`, line 707)

Document or section may not begin with a transition.

Flow Control

- If, elif, else
- Loops
  - for/for each
  - range() and xrange()
  - while
  - break
  - continue
  - pass
  - else in loops

IF...ELIF...ELSE Statements

```
a = 100
if a > 100:
    print 'a is greater than 100'
elif a < 100:
    print 'a is less than 100'
else:
    print 'a is 100'
```

Note

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.

WHILE LOOPS

```
count = 0
while count < 10:
    print 'count: ' + str(count)
    count += 1
```

WHILE LOOPS...ELSE??

```
count = 0
while count < 10:
    print 'count: ' + str(count)
    count += 1
```

```
else:
    print 'count is now: ' + str(count)
```

> **Note**
>
> the else statement is executed when the while loop condition becomes false

---

FOR LOOPS

For loop in C

```
char *my_string = "Python";
for (int i = 0; i < strlen(my_string); i++)
{
    printf("%c\n", my_string[i]);
}
```

For loop in Python

```
my_string = "Python"
for i in range(len(my_string)):
    print my_string[i]
```

> **Note**
>
> Not really pythonic, this is kind of like writing python that looks like c

---

FOR LOOPS cont'd

This is a better for loop in pyhon

```
my_string = "Python"
for letter in my_string:
    print letter
```

> **Note**
>
> read this as for every letter in python

FOR LOOPS...ELSE??

```
my_string = "Python"
for letter in my_string:
    print letter
else:
    print "done"
```

> **Note**
>
> the else is executed when the loop has exhausted iterating the list

---

LAB: Operation Flow Control

> **System Message: ERROR/3 (`combined.rst`, line 833)**
>
> Document or section may not begin with a transition.

Input and Output

- Print
- Str + str style
- Printf style

Printing

```
print 'hello'
print 1 + 1
print 'Hello' + ' World'
print 'I am' + 1337 ?
print 'I am', 1337  ?
```

Printing cont'd

```
print "%s World!" % "Hello"
print "%s %s!" % ("Hello", "World")
print "{} {}!".format("Hello", "World")
print "{1} {0}!".format("Hello", "World") ?
```

> **Note**
>
> follows c sprintf style

Printing: Format Symbols

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

LAB: Using Python’s print

> **System Message: ERROR/3 (`combined.rst`, line 882)**
>
> Document or section may not begin with a transition.

Files

- Common Operations
- Binary vs “Python text”
- Modes

Open a file

```
f = open('filename', 'mode')

f = open('file.txt', 'wb')
data = f.read()
f.close()
```

> **Note**
>
> so what's with the b? Linux everything is a file, windows treats binaries and files differently

---

File Operations

```
f.write(str)        #write str to file
f.writelines(str)   #write str to file
f.read(sz)          #read size amount
f.readline()        #read next line
f.seek(offset)      #move file pointer to offset
f.tell()            #current file position
f.truncate(sz)      #trucate the file sz bytes
f.close()           #close file handle
```

Open a file cont'd

```
with open('filename', 'mode') as f:
    data = f.read()
```

> **Note**
>
> no need to close handle as that's handled by the 'with' construct

---

File Modes

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

LAB: Working with files

System Message: ERROR/3 (`combined.rst`, line 949)

Document or section may not begin with a transition.

User Functions

System Message: ERROR/3 (`combined.rst`, line 954)

Document or section may not begin with a transition.

- Parameters
- Scope
- Returning value/multiple values
- Pass by reference only. Be careful how you treat parameter values
- Default arguments
- Command line arguments
- Keyword arguments (**kwargs)
- Variable length arguments

Note

block of organized reusable code

Functions

```
def function_name(parameters):
    do stuff
    return stuff
```

Note

parameters are passed by references

Functions cont'd

```
def add_subtract(a = 1, b = 1):
    return (a+b, a-b)

x, y = add_subtract(3, 2)
x, y = add_subtract(3, )
```

Note

default arguments, multiple return and assign

Function Scope

Three levels:

- local
- global/module
- builtin

Function Scope cont'd

```
a = 1

def global_var_read():
    print a

def local_var_assign():
    a = 2
    return a+a

def global_var_write():
    a += 3
    return a
```

Note

invalid global write

Function Scope cont'd

```
a = 1

def global_var_write():
    global a
    a += 3
```

Note

invalid global write

Function Scope cont'd

```
a = []

def global_var_write():
    a.append(1)
```

> **Note**
>
> works if a is mutable

Keyword Arguments

- *args is used to pass a non-keyworded, variable-length argument list
- **kwargs is used to pass a keyworded, variable-length argument list

```
def my_func1(*args):
    for arg in args:
        print 'arg: ', arg

def my_func2(**kwargs):
    for key, value in kwargs.iteritems():
        print key, value

my_func1('two', 3, 'four', 5)
my_func2(a=12, b = 'abc')
```

> **Note**
>
> arguments passed in are placed in a dictionary

Cmdline Arguments - old

```
import sys

args = str(sys.argv)

print sys.argv[0]
for i in range(len(sys.argv)):
    print 'args[%d]: %s' % (i, sys.argv[i])
```

Cmdline Arguments - new

```
import argparse

parser = argparse.ArgumentParser(description='This is a demo')
parser.add_argument('-i','--input',help='Input arg', required=True)
args = parser.parse_args()

print args.input
```

LAB: Working with functions

> **System Message: ERROR/3 (`combined.rst`, line 1125)**
>
> Document or section may not begin with a transition.

More Python Stuff

- Lambdas
- Comprehensions
- Closures

- Generators, Iterators, Yield

Anonymous Functions

- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

> **Note**
>
> http://www.tutorialspoint.com/python/python_functions.htm

Lambdas

```
my_list = range(26)
print filter(lambda x: x % 2 == 0, my_list)

g = lambda x,y: x > y
g(1,2)
g(2,1)
```

> **Note**
>
> what does this do? prints even numbers from 0 to 25

LAB: Working with lambdas

> **System Message: ERROR/3 (`combined.rst`, line 1173)**
>
> Document or section may not begin with a transition.

List Comprehensions

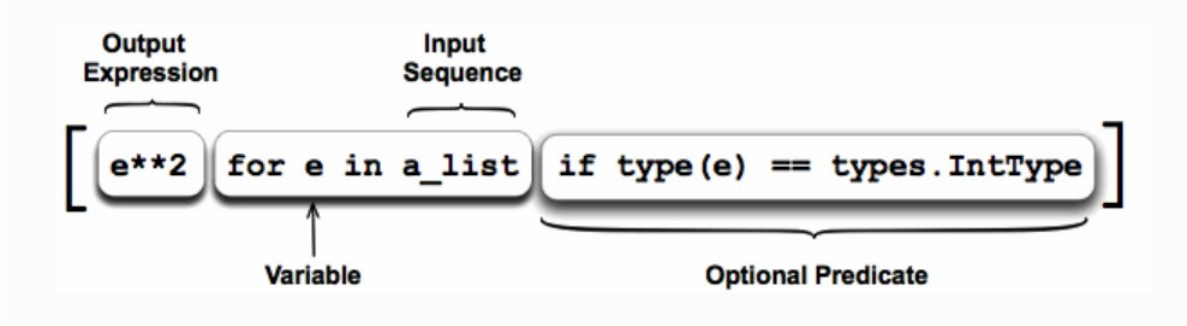
```
a_list = [1, 2, 3, 4, 5]
def square_list(a_list):
    a = []
    for item in a_list:
        a.append(item*item)
    return a

a_list = [1, 2, 3, 4, 5]
def square_list(a_list):
    for i in range(len(a_list)):
        a_list[i] *= a_list[i]
```

List Comprehensions cont'd

```
a_list = [1, 2, 3, 4, 5]
def square_list(a_list):
    return [x*x for x in a_list]
```

List Comprehension construct



List Comprehensions cont'd

```
a_list = [1, 2, 3, 4, 5]
def square_list(a_list):
    return [x*x for x in a_list if x % 2 == 0]
```

Closures

A CLOSURE is a function object that remembers values in enclosing scopes regardless of whether those scopes are still present in memory. If you have ever written a function that returned another function, you probably may have used closures even without knowing about them.

Closures

```
def generate_power_func(n):
    def nth_powah(x):
        return x**n
    return nth_powah
```

Generators

```
def yrange(n):
    i = 0
    while i < n:
        yield i
        i += 1
```

Note

when functions return all state is lost, a yield preserves local state and returns a generator object

System Message: WARNING/2 (`combined.rst`, line 1259)

Explicit markup ends without a blank line; unexpected unindent.

Python Exceptions

System Message: ERROR/3 (`combined.rst`, line 1264)

Document or section may not begin with a transition.

Exception Handling

- Try/except/finally/else
- Raise

- Multiple exceptions

> **Note**
>
> try statement begins an exception handling block raise triggers exceptions except handles the exception

---

LAB: Exceptions lab

> **System Message: ERROR/3 (`combined.rst`, line 1283)**
>
> Document or section may not begin with a transition.

Python Modules

> **System Message: ERROR/3 (`combined.rst`, line 1288)**
>
> Document or section may not begin with a transition.

Modules

- Single file (.py, .pyc, or .pyo)
- Reusable Code for use across other scripts

Module Structure

```
# triangle.py

divisor_count_to_find = 500

def triangle_number(n):
    return n*(n+1)/2

def divisors(tn):
    counter = 0
    n_sqrt = int(pow(tn, 0.5))
    for i in xrange(1, n_sqrt+1):
        if tn%i == 0:
            counter += 2
        if n_sqrt * n_sqrt == tn:
            counter -= 1
    return counter

def start():
    start_number = 1
    div_numbers = 0
    while (div_numbers < divisor_count_to_find):
        tn = triangle_number(start_number)
        div_numbers = divisors(tn)
        start_number += 1

    print div_numbers
    print tn

if __name__ == '__main__':
    start()
```

Module Use

```
>> import triangle
>> triangle.start() # 576, 76776500
>> triangle.triangle_number(10)  # 55
>> triangle.divisors(10) # 4
>> triangle.divisor_count_to_find = 100
>> triangle.start() # 112, 73920
```

> **Note**

when a package is imported, code not in a function is ran, function definitions are added to the namespace and variables are set.

Importing

Ways to import

- import triangle
  - triangle.<method>
- import triangle as tri
  - tri.<method>
- from triangle import *
  - <method>

More on Importing

```
Importing

    from module import *

You've probably seen this "wild card" form of the import
statement. You may even like it. Don't use it.

To adapt a well-known exchange:
    (Exterior Dagobah, jungle, swamp, and mist.)

    LUKE: Is from module import * better than explicit
    imports?

    YODA: No, not better. Quicker, easier, more
    seductive.

    LUKE: But how will I know why explicit imports are
    better than the wild-card form?

    YODA: Know you will when your code you try to read
    six months from now.

Wild-card imports are from the dark side of Python.
Never!

The from module import * wild-card style leads to
namespace pollution. You'll get things in your local
namespace that you didn't expect to get. You may see
imported names obscuring module-defined local names.
You won't be able to figure out where certain names
come from. Although a convenient shortcut, this should
not be in production code.
```

Note

http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#importing

---

Packages

- Collection of modules
- Can also include addtional packages
- A module named "__init__.py" is required to treat the directory as a package
  - contains initialization code

Package Structure

```
math_stuff/             #top directory
    __init__.py         #special module
    fib.py              #fib module
    triangle.py         #triangle module
    calculator.py       #calculator module
    more_math_stuff/    #additional package
        __init__.py
        other_modules.py
```

Package Structure cont'd

```
__init__.py
    math_stuff.__all__
    ['fib','triangle','calculator','more_math_stuff']
```

Package Structure cont'd

Useful commands

- dir()
- help()

Note: if a directory has a package and module named the same, the package will be loaded instead of the module

---

Python Classes

System Message: ERROR/3 (`combined.rst`, line 1465)

Document or section may not begin with a transition.

User Classes

- Defining a class
- __init__ (Constructor)
- __del__ (Destructor) optional
- Variables defined under class are shared among all instances
- Variables defined in init are only for a single instance
- When creating functions, the first parameter must be self. (similar to “this” in other languages)
- A “private” member can be created by prepend a double underscore (only prepend) to the name. It will not be visible outside of that class
- Inheritance, Polymorphism, double inheritance(?) are all supported

---

Using Classes

- Create instance of class, do not need to pass “self” even though it is in the __init__ definition
- Var.attrib = value is the common way to interact(dot notation)
- Getattr, hasattr, setattr, delattr exist for manipulating attributes as well.
- Some Built in class attributes: __dict__, __doc__, __name__, __module__, __bases__
- Some Overloadable base class attributes __init__, __del__, __repr__, __str__, - __cmp__
- Dir(), type(), isinstance() built in functions

User Classes

```
class ClassName(object):
    def __init__(self, arg):
        super(ClassName, self).__init__()
        self.arg = arg
```

Create a child class

Python OO

Unlike other languages, python has some gotchas. Python doesn't really have public, protected, or private

A single '_' indicates internal usage. '__' hides attribute name

LAB: Working with classes

System Message: ERROR/3 (`combined.rst`, line 1519)

Document or section may not begin with a transition.

Python Exceptions

Exception Handling

- Try/except/finally/else
- Raise
- Multiple exceptions

> **Note**
>
> try statement begins an exception handling block raise triggers exceptions except handles the exception are resource intensive

---

Exception Handling

```
try:
    <statements>
except <name>:
    <statements>
except <name> as <data>:
    <statements>
else:
    <statements>

try:
    <statements>
finally:
    <statements>
```

Exception Structure (Class)

```
class CustomError(Exception):
    def __init__(self, msg)
        self.msg = msg

    def __str__(self):
        return "your error is {}".format(self.msg)
```

Exception Structure (Class) ==========s====

```
try:
    doStuff
except  CustomError as stuff:
    print stuff

def doStuff(danger):
    if danger == True:
        raise CustomError("Whoa don't do that!")
```

LAB: Exceptions lab

Python Threading

> **System Message: ERROR/3 (`combined.rst`, line 1602)**
>
> Document or section may not begin with a transition.

Threading

- Basic usage
- Limitations (GIL)
    - The global interpreter lock, or GIL, is a mutex that prevents multiple native threads from executing Python bytecodes at once. This lock is necessary mainly because CPython's memory management is not thread-safe

---

Creation

```
threading.Thread(target=func, args=args, kwargs=kwargs)
<thread>.start()
<thread>.join()
```

Example

```
g_counter = 0
g_lock = threading.Lock()
def inc_counter(times=50, lock=False):
    global g_counter
    for i in xrange(times):
        if lock:
            g_lock.acquire()
            g_counter += 1
            g_lock.release()
        else:
            g_counter += 1
```

Example cont'd

```
def run_threads(num_threads = 10, lock = False):
    global g_counter
    global_counter = 0
    threads = [threading.Thread(target=inc_counter, \
    args=args, kwargs=kwargs) for i in xrange(num_threads)]
    for thread in threads:
        thread.start()
    for thread in threads:
        thread.join()
```

Shared Resources

- One important issue when using threads is to avoid conflicts when more than one thread needs to access a single variable or other resource. If you're not careful, overlapping accesses or modifications from multiple threads may cause all kinds of problems, and what's worse, those problems have a tendency of appearing only under heavy load.
- If you call this function from more than one thread, you'll find that the counter isn't necessarily accurate. It works in most cases, but sometimes misses one or more items. The reason for this is that the increment operation is actually executed in three steps; first,the interpreter fetches the current value of the counter, then it calculates the new value, and finally, it writes the new value back to the variable.
- If another thread gets control after the current thread has fetched the variable, it may fetch the variable, increment it, and write it back, before the current thread does the same thing. And since they're both seeing the same original value, only one item will be accounted for.
- Another common problem is access to incomplete or inconsistent state, which can happen if one thread is initializing or updating some non-trivial data structure, and another thread attempts to read the structure while it's being updated.

> **Note**
>
> http://effbot.org/zone/thread-synchronization.htm

Atomic Operations

- The simplest way to synchronize access to shared variables or other resources is to rely on atomic operations in the interpreter. An atomic operation is an operation that is carried out in a single execution step, without any chance that another thread gets control.
- Use a lock; locks are typically used to synchronize access to a shared resource. For each shared resource, create a Lock object. When you need to access the resource, call acquire to hold the lock (this will wait for the lock to be released, if necessary), and call release to release it.

LAB: Threading lab

> **System Message: ERROR/3 (`combined.rst`, line 1688)**
>
> Document or section may not begin with a transition.

Python ctypes

System Message: ERROR/3 (combined.rst, line 1693)

Document or section may not begin with a transition.

---

ctypes

- Interface with C libraries
- Faster
- Cross Platform
- GIL release

Data types

ctypes type	C type	Pyth
c_bool	_Bool	bool (1)
c_char	char	1-characte
c_wchar	wchar_t	1-characte
c_byte	char	int/long
c_ubyte	unsigned char	int/long
c_short	short	int/long
c_ushort	unsigned short	int/long
c_int	int	int/long
c_uint	unsigned int	int/long
c_long	long	int/long
c_ulong	unsigned long	int/long
c_longlong	__int64 or long long	int/long
c_ulonglong	unsigned __int64 or unsigned long long	int/long
c_float	float	float
c_double	double	float
c_longdouble	long double	float
c_char_p	char * (NUL terminated)	string or No
c_wchar_p	wchar_t * (NUL terminated)	unicode or
c_void_p	void *	int/long or

Calling a function

- define argument types (argtypes)
- define return type (restype)

```
calc = ctypes.CDLL("MyCalc.dll")
addition = calc.addition
```

```
calc.restype = ctypes.c_int
calc.argtypes = [ctypes.c_int, ctypes.c_int]
addition(1,2)
```

---

structures

```
class P_Struct(ctypes.Structure):
    _fields_ = [("field_1", ctypes.c_int),
                ("field_2", ctypes.char_p)]

my_struct = P_Struct(1, "Hello World")
pointer_my_struct = ctypes.pointer(my_struct)
```

Buffers

- Remember python strings are IMMUTABLE
- ctypes.create_string_buffer("<str>")

LAB: Working with ctypes

System Message: ERROR/3 (combined.rst, line 1751)

Document or section may not begin with a transition.

Python Stuff

System Message: ERROR/3 (combined.rst, line 1756)

Document or section may not begin with a transition.

Regex

- Regex characters, special characters
- Matches
- Groups
- Substitution
- “compiling” regex

Python re

- re.compile: compile re expression for repeated use
- re.match: apply pattern to start of string, return match or None
- re.search: scan through string, return match or None

---

regex

```
import re
search_str = "This is a string to search"
re_search = re.compile("string")
matched_obj = re_search.search(search_str)
```

LAB: Working with regex

System Message: ERROR/3 (combined.rst, line 1793)

Document or section may not begin with a transition.

Libraries and Modules

- os, socket, subprocess, hashlib, shutil, math, json
- struct
- 3rd party
- Paramiko
- Pip
- Hitchhikers guide

Working with os, sys, subprocess

- sys module
    - contains system level info
    - sys.path, sys.argv, sys.modules
- os module
    - interact with the os dependent functionality
    - os.path, os.walk, os.system, os.stat
- subprocess module
    - spawn processes, stdin/stdout
    - subprocess.Popen()s

Hashlib

```
import hashlib
m = hashlib.md5()
m.update(some data)
print m.digest()
print m.hexdigest()
```

Struct

This module performs conversions between Python values and C structs represented as Python strings.

```
import struct
data = struct.pack('hhl', 1, 2, 3)
print data
data = struct.unpack('hhl', data)
print data
```

Lab

- Using the os module, create a program that will do a recursive directory search and create a dictionary of the results, using the dir path as the key and files in a list
- Use the subprocess module to create a program to list the processes running and pipe to a file, start and kill an instance of calc.exe or any other program.
- Use structs to pack and unpack a name, age, location.
- Using hashlib, write a program that will find duplicate files in a given directory.

Serialization

- marshalling, flattening
- Json
- Pickle

Pickle

```
import pickle

data1 = {'a': [1, 2.0, 3, 4+6j],
         'b': ('string', u'Unicode string'),
         'c': None}

selfref_list = [1, 2, 3]
selfref_list.append(selfref_list)

output = open('data.pkl', 'wb')

# Pickle dictionary using protocol 0.
pickle.dump(data1, output)

# Pickle the list using the highest protocol available.
pickle.dump(selfref_list, output, -1)
```

```
output.close()
```

Pickle cont'd

```
import pprint, pickle

pkl_file = open('data.pkl', 'rb')

data1 = pickle.load(pkl_file) pprint.pprint(data1)

data2 = pickle.load(pkl_file) pprint.pprint(data2)

pkl_file.close()
```

Working with serialization

System Message: ERROR/3 (`combined.rst`, line 1920)

Document or section may not begin with a transition.

Socket Programming

- Socket API
- TCP, UDP,
- RAW sockets (Linux)