

What Led to This?

- Python 2.7 End of Life is January 1, 2020.
- I have several custom tools built over the last 5 years.
- Decided it was (well past) time to start testing python 3 compatibility.
- During testing I found several little caveats and porting tricks.
- I thought would be handy to share here.
- Plus Brandi needed a backup speaker. :-P

So What's the Deal with the Incompatibility?

- Python 2.7 was a lot more relaxed with typing.
 - This was a selling point originally.
 - But now they are starting see some of the issues with loose typing.
- They wanted to re-factor some code changes as well, change naming conventions.
- New features, changes to features, and behavior changes.
- Unicode is also a thing it seems...

So What Kinda New Changes are There?

- Print is now exclusively a function, and not a keyword statement.
 - Print() also got some upgrades.
- Division will now always return a float point instead of an int.
- Several functions return iterators now instead of list.
- raw_input() has been renamed to input().



- Unicode is the defacto now.
- ints are technically longs now.
- Formatting system changes as well
 - and seems to keep changing across minor revisions *facepalm*

How Has the Community Taken to This?

- Poorly...
- Either went all in on 3.x or staying on 2.7
- Basically it fractured the user base.
- The devs have taken note of this and admitted this roll out was poorly executed.
- Apologized, wrote a letter explaining why these changes are going through.
- Stated they will never do this extreme of a change between version again.

What Should I Do In This Case?

- EOL is approaching fast in less than a year.
- We are at a cross road of three options:
 - Switch entirely to a new language.
 - Stay on the older, stagnating, no longer supported version.
 - Likely to become buggy down the road.
 Possible security issues down the road.
 - Or convert over to the new version of python.

Which Did I Choose?

- The thought of converting to C had crossed my mind...
 - Pro: Rarely changes unlike high level languages
 - Pro: Truly cross architecture. Lots of embedded systems don't ship with python.
 - Pro: Binaries can be shipped statically.
 - Pro: Really easy to package as DEB files and push to my machines.

Which Did I Choose?

- The thought of converting to C had crossed my mind...
 - Pro: Matured community and documentation.
 - Con: Really big porting effort. (REALLY BIG CON!)
 - Con: Losing all the existing modules.
 - Con: Easier to create security bugs.



- Staying on an unsupported version is simply out of the question.
- Guess we'll switch to version 3 ¯_(ッ)_/¯
 - To do that we will need to:
 - Test what works as is, and what breaks.
 - Figure out workarounds, preferably that work on both versions.

What Issues Did I Run into Specifically During Testing?

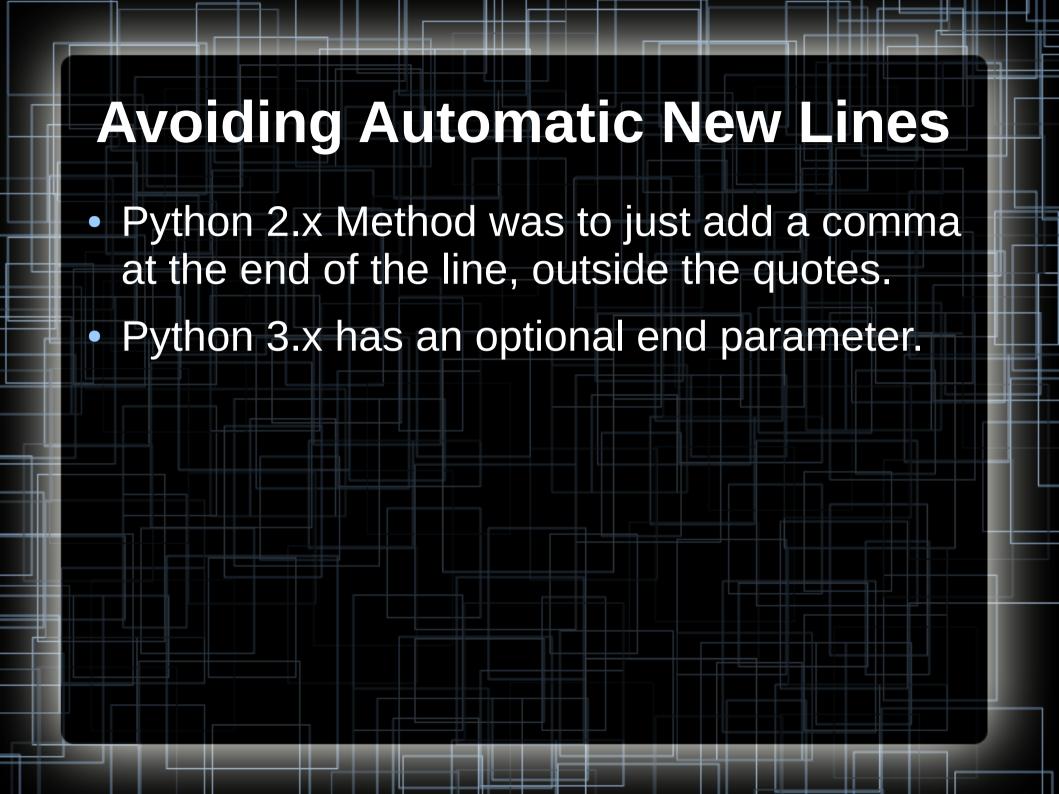
- Primarily it fell into one of two issues:
- Syntax changes
 - print needs to be a function.
 - Formatting strings needed changes.
 - Renamed core functions needed updates.
- Libraries
 - Some had Python 3 ports & some didn't.
 - Sometimes there were differences between them.

Can We Make the Code Friendly Between the Two Version?

- In most cases, yes.
- Python created a few modules (__future___. builtins, etc)
 - Designed to abstract 3.x conventions in older 2.x versions of python.
- Modules usually tend to be the biggest challenges here.



- In python 2, print was both a function and a keyword.
- In Python 3, it is now only a function.
- Valid in python 2, but not 3:
 - print "Hello World"



Python 2 Method

print "Don't finish this line yet...",

print "Okay Newline now!"

Works with 2... but not 3.

```
print "Hello World!"

$ python2 ./no_new_line_python_2.py
Don't finish this line yet... Okay Newline now!
Hello World!

$ python3 ./no_new_line_python_2.py
File "./no_new_line_python_2.py", line 1
    print "Don't finish this line yet...",

SyntaxError: Missing parentheses in call to 'print'
```

Python 3 Method

Works with 3... but not 2

Welcome to the future

- So the past examples show a "damned if you do, damned if you don't" type of use case.
- Future module can help here.

Easy, clean, reliable Python 2/3 compatibility

python-future is the missing compatibility layer between Python 2 and Python 3. It allows you to use a single, clean Python 3.x-compatible codebase to support both Python 2 and Python 3 with minimal overhead.

Making Compatible Between 2.7 and 3 using __future___

 Using Future, the python 3 code works on both 2.x and 3.x.

```
from __future__ import print_function

print("Don't finish this line yet...", end=" ")
print("Okay Newline now!")
print("Hello World!")

$ python2 ./no_new_line_python_universal.py
Don't finish this line yet... Okay Newline now!
Hello World!
$ python3 ./no_new_line_python_universal.py
Don't finish this line yet... Okay Newline now!
Hello World!
$
```

Syntax Issues #2: raw_input() vs input()

- raw_input() was renamed to input().
- Input() is not the same function in python2 as it is in python3.
- raw_input() is not recognized by python3.
- Another damned if you do, damned if you don't...
- Unless you use "from builtins import input"

Python 2 Method name = raw input("What's Your Name: print("Hello, {0:s}".format(name)) \$ python2 ./get input python2.py What's Your Name: Travis Hello, Travis \$ python3 ./get input python2.py Traceback (most recent call last): File "./get input python2.py", line 1, in <module> name = raw input("What's Your Name: ") NameError: name 'raw input' is not defined

Python 3 Method

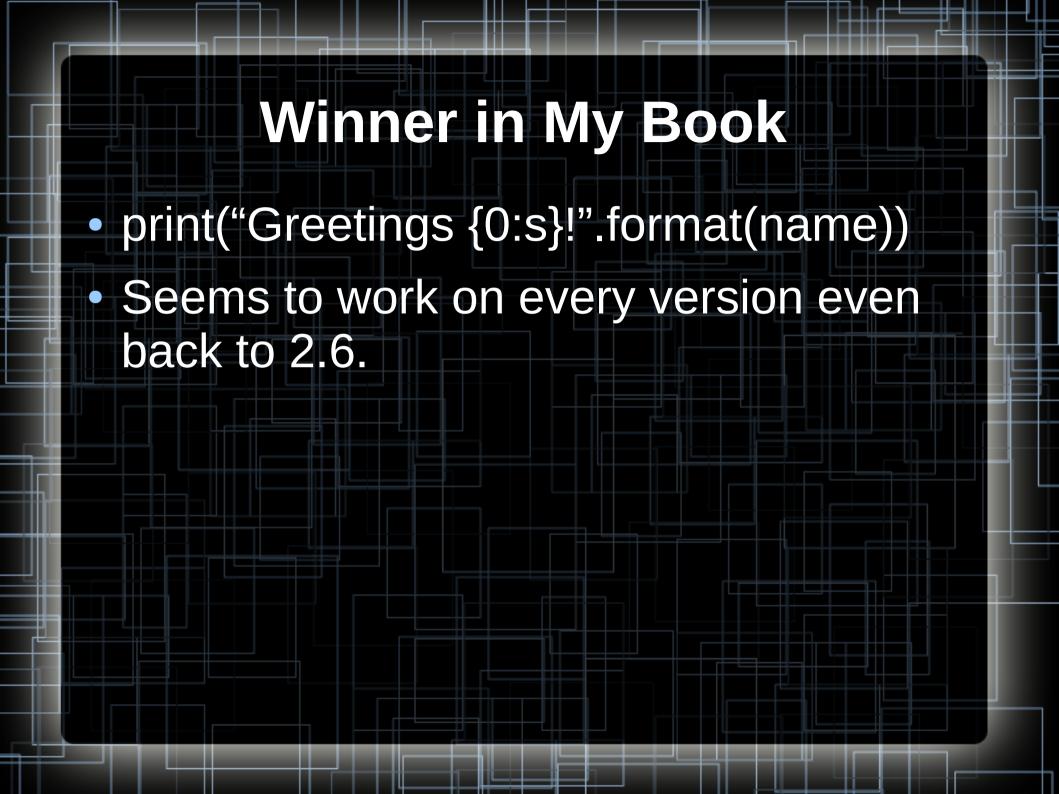
```
name = input("What's Your Name: ")
print("Hello, {0:s}".format(name))
```

```
$ python3 ./get_input_python3.py
What's Your Name: Travis
Hello, Travis
$ python2 ./get_input_python3.py
What's Your Name: Travis
Traceback (most recent call last):
   File "./get_input_python3.py", line 1, in <module>
        name = input("What's Your Name: ")
   File "<string>", line 1, in <module>
NameError: name 'Travis' is not defined
$
```

Universal Method from builtins import input name = input("What's Your Name: ") print("Hello, {0:s}".format(name)) \$ python2 ./get input universal.py What's Your Name: Travis Hello, Travis \$ python3 ./get input universal.py What's Your Name: Travis Hello, Travis

Syntax Issues #3: Line Formatting and Concatenation

- HO-LEE SHIT! Does python have a lot of ways of doing this! These are all valid, and not always cross version compatible. (Sometimes not even compatible across minor revisions)
- print("Greetings " + name + "!")
- print("Greetings %s!" % (name))
- print("Greetings {}!".format(name))
- print("Greetings {0}!".format(name))
- print("Greetings {0:s}!".format(name))
- print("Greetings {var}!".format(var=name))
- print(f"Greetings {name}!")



Syntax Issues #4: Int vs Float Returns in Division

- One of the smartest moves in Python 3 in my opinion.
- Division in Python 2 returns an int on ints and float if at least one number was a float

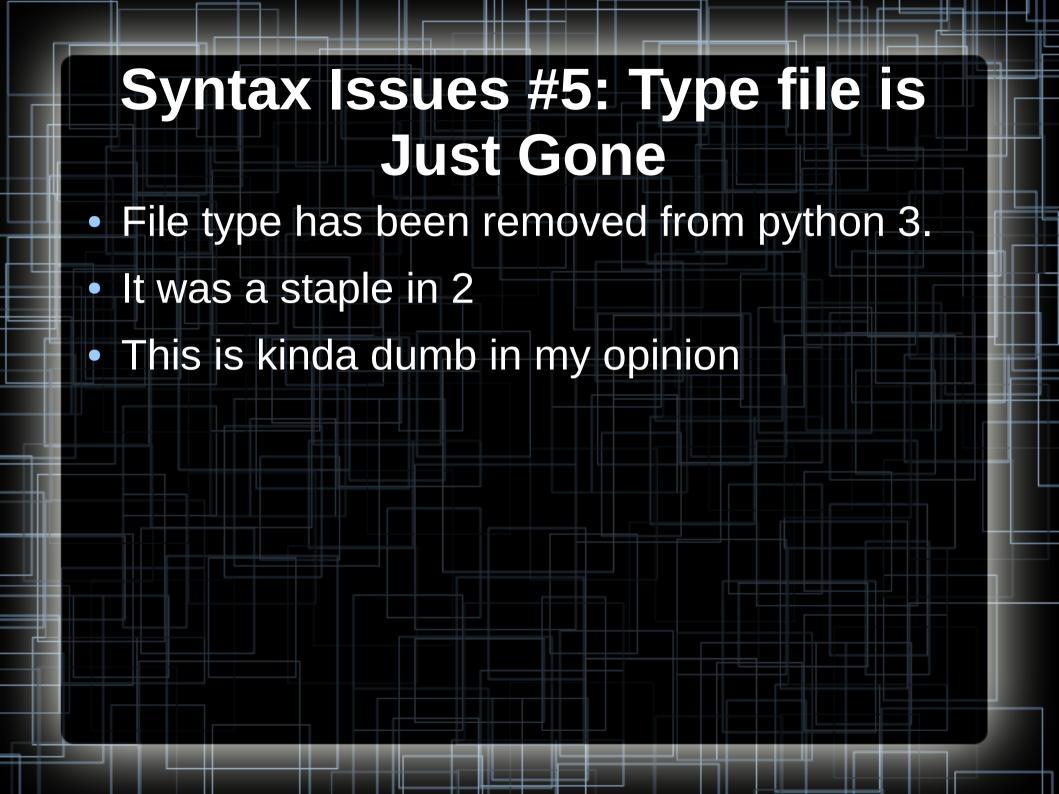
$$-5/2=2$$

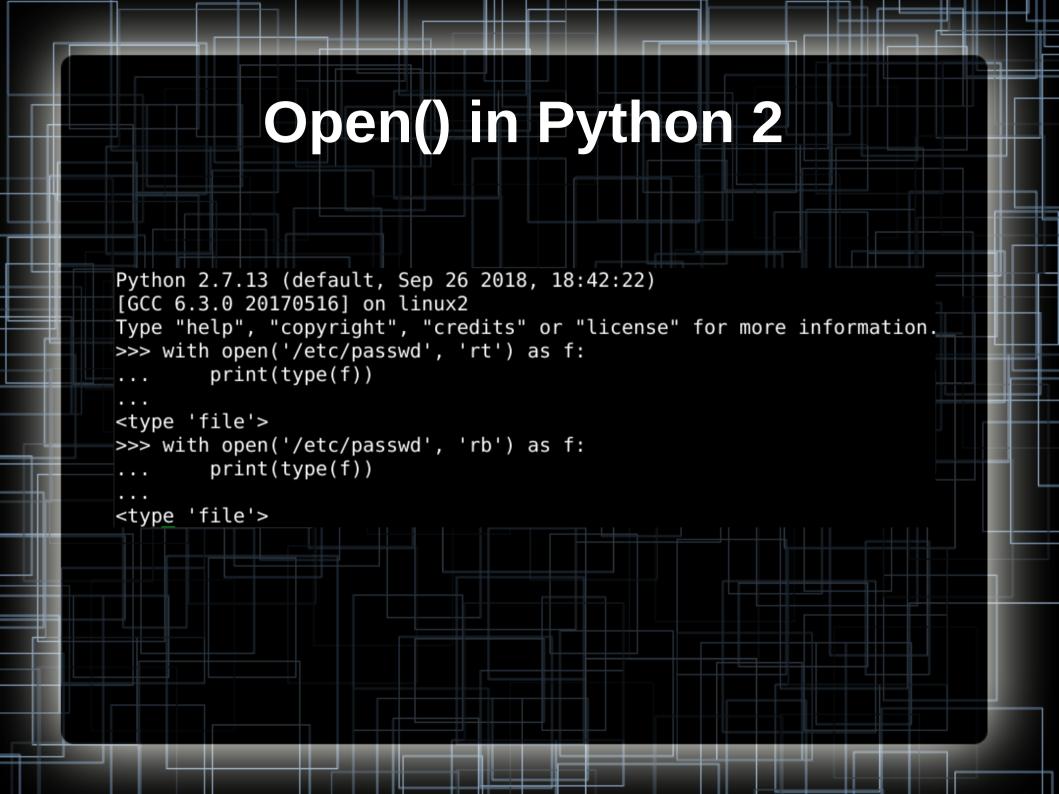
$$-5.0/2 = 2.5$$

 Division in Python 3 returns a floating point number... ALWAYS!!!

$$-5/2 = 2.5$$

Can Be Made Universal from future import division $print("5 / 2 = {0:.01f}".format(5/2))$ \$ python2 ./division universal.py 5 / 2 = 2.5python3 ./division universal.py 5 / 2 = 2.5

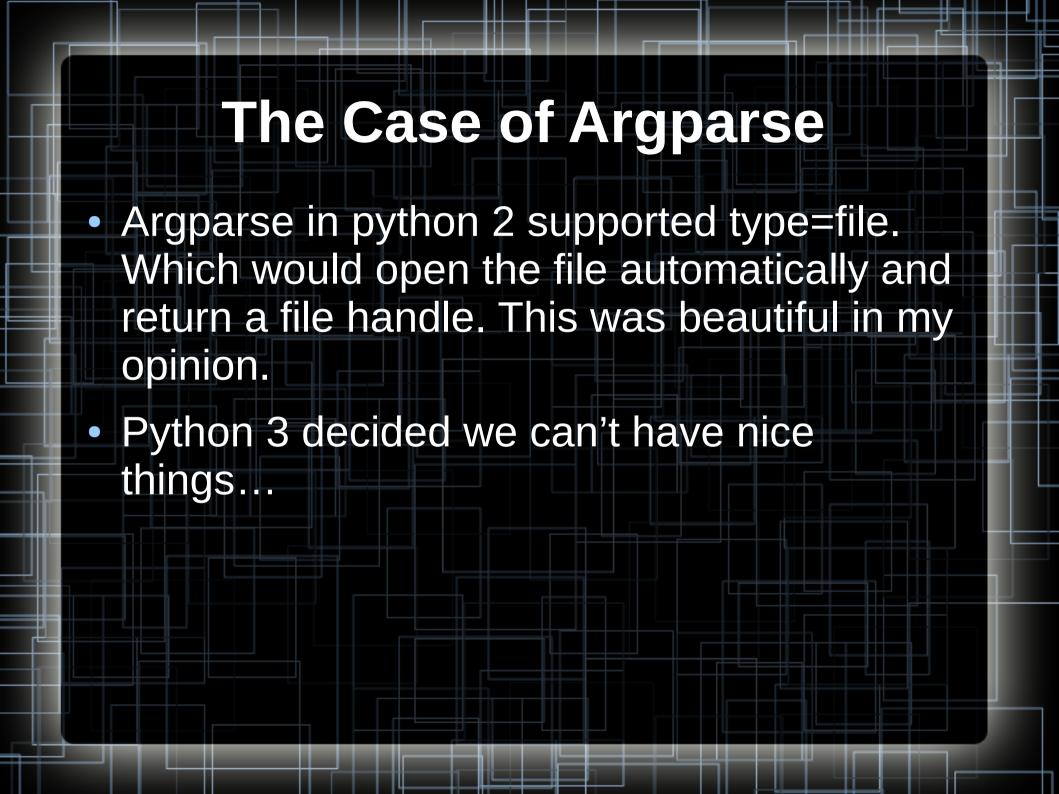




Open() in Python 3 Python 3.5.3 (default, Sep 27 2018, 17:25:39) [GCC 6.3.0 20170516] on linux Type "help", "copyright", "credits" or "license" for more information. >>> with open('/etc/passwd', 'rt') as f: print(type(f)) <class ' io.TextIOWrapper'> >>> with open('/etc/passwd', 'rb') as f: print(type(f)) <class ' io.BufferedReader'>



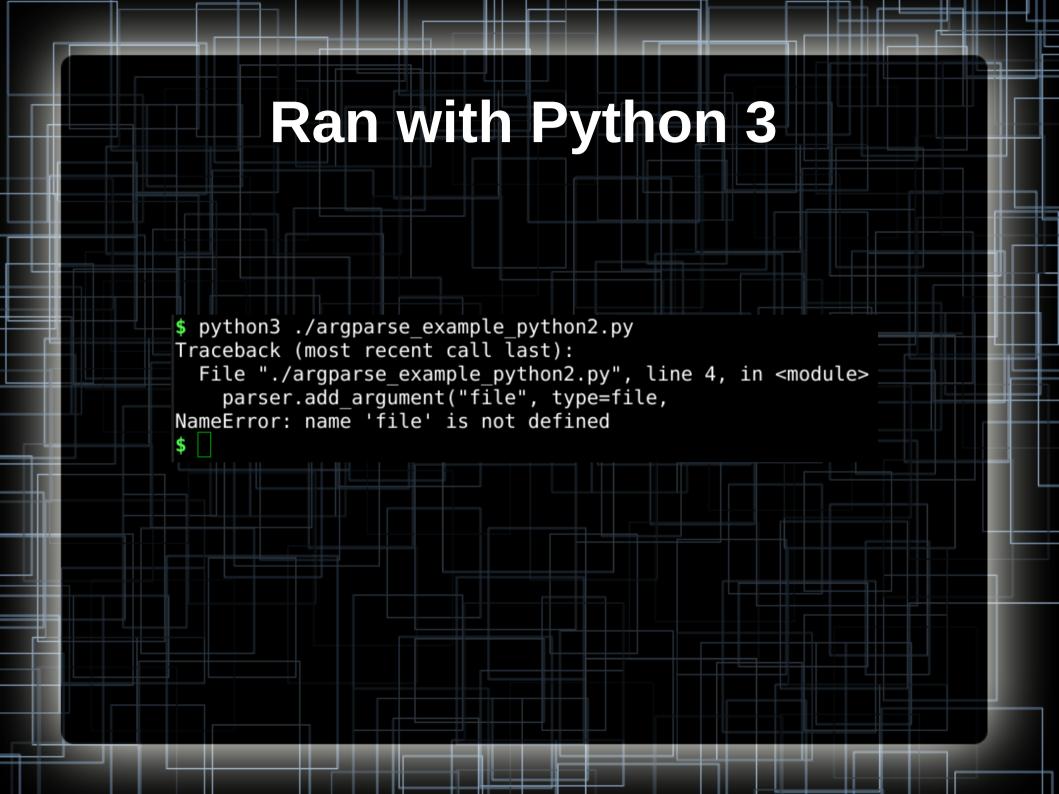
- In C, you open a file, get int as a file handle back. This works well enough...
- Why Multiple Classes?
- Anywho, main places I've seen this issue is Argparse and also with reads since the whole str vs bytes non-sense (covered later)

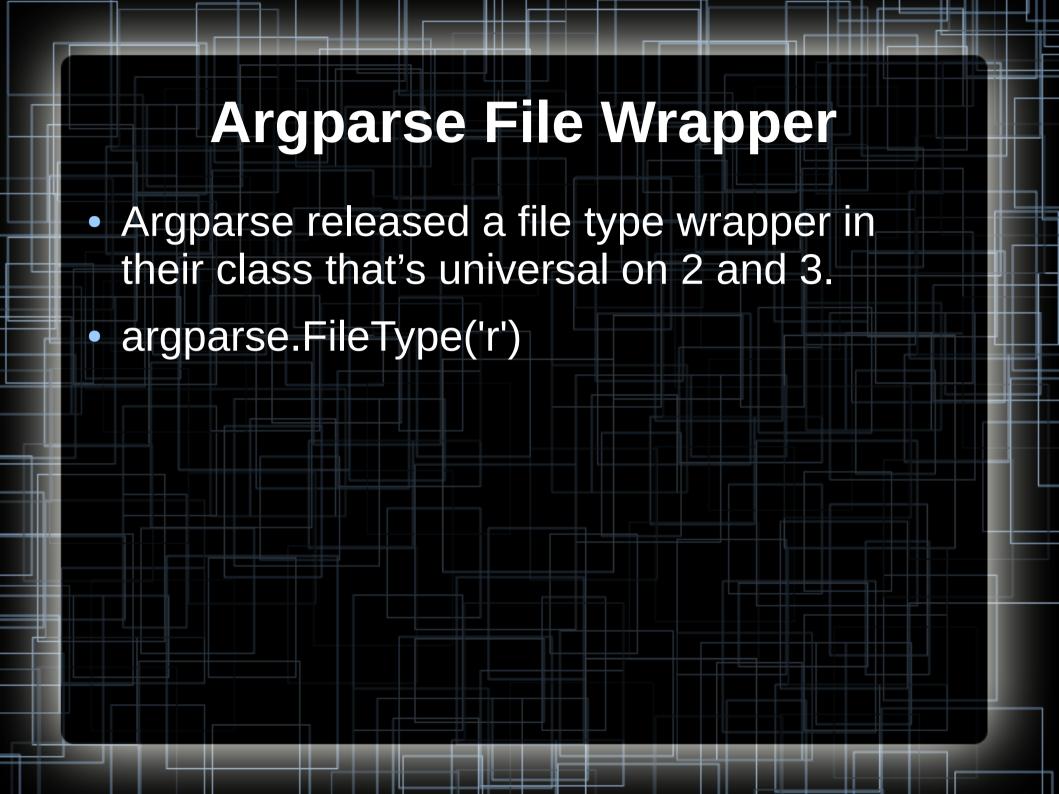


Example Code import sys import argparse parser = argparse.ArgumentParser(description="Check if a file exist") parser.add argument("file", type=file, help="check if file is valid.") if len(sys.argv) == 1: parser.print_help(sys.stderr) sys.exit(1)args = parser.parse args() print(type(args.file)) print("Seems to exist!")

Ran with Python 2

```
$ python2 argparse example python2.py
usage: argparse example python2.py [-h] file
Check if a file exist
positional arguments:
  file
              check if file is valid.
optional arguments:
  -h, --help show this help message and exit
$ python2 argparse example python2.py /etc/passwd
<type 'file'>
Seems to exist!
$ python2 argparse example python2.py /etc/passwwd
Traceback (most recent call last):
  File "argparse example python2.py", line 9, in <module>
    args = parser.parse args()
  File "/usr/lib/python2.7/argparse.py", line 1701, in parse args
    args, argv = self.parse known args(args, namespace)
  File "/usr/lib/python2.7/argparse.py", line 1733, in parse known args
    namespace, args = self. parse known args(args, namespace)
  File "/usr/lib/python2.7/argparse.py", line 1942, in parse known args
    stop index = consume positionals(start index)
  File "/usr/lib/python2.7/argparse.py", line 1898, in consume positionals
    take action(action, args)
  File "/usr/lib/python2.7/argparse.py", line 1791, in take action
    argument values = self. get values(action, argument strings)
  File "/usr/lib/python2.7/argparse.py", line 2231, in get values
   value = self. get value(action, arg string)
  File "/usr/lib/python2.7/argparse.py", line 2260, in get value
    result = type func(arg string)
IOError: [Errno 2] No such file or directory: '/etc/passwwd'
```





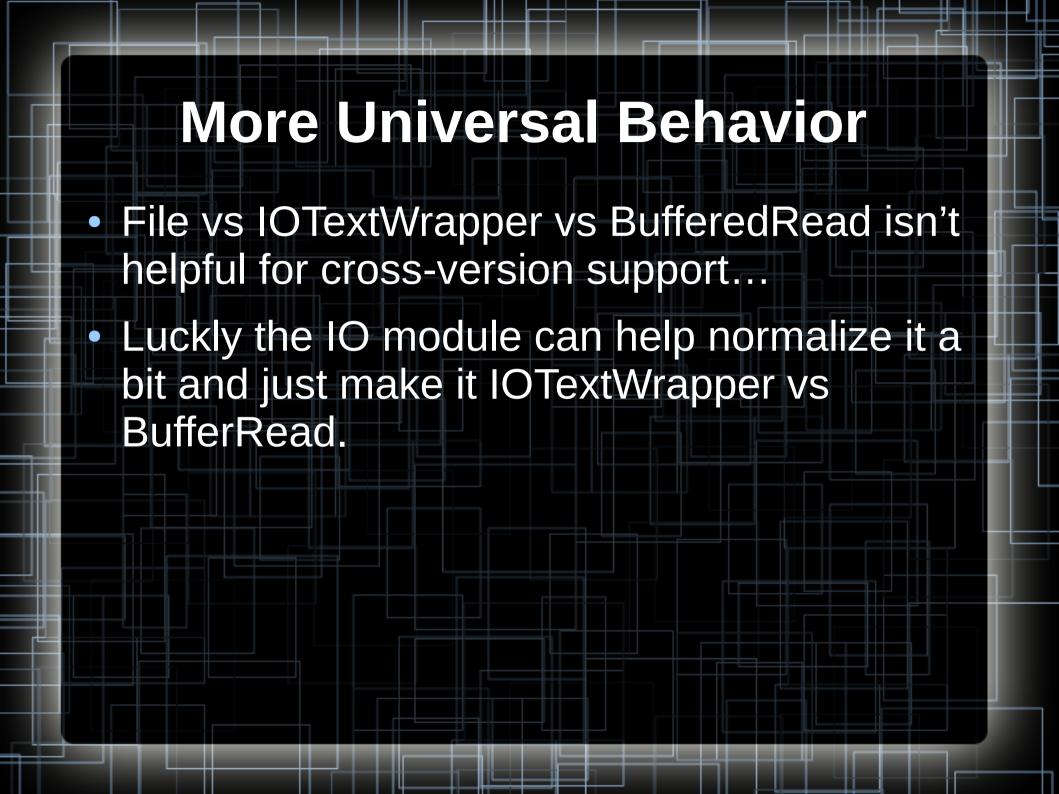
```
Example Code
import sys
import argparse
parser = argparse.ArgumentParser(description="Check if a file exist")
parser.add argument("file", type=argparse.FileType('r'),
               help="check if file is valid.")
if len(sys.argv) == 1:
       parser.print help(sys.stderr)
       sys.exit(1)
args = parser.parse_args()
print(type(args.file))
print("Seems to exist!")
```

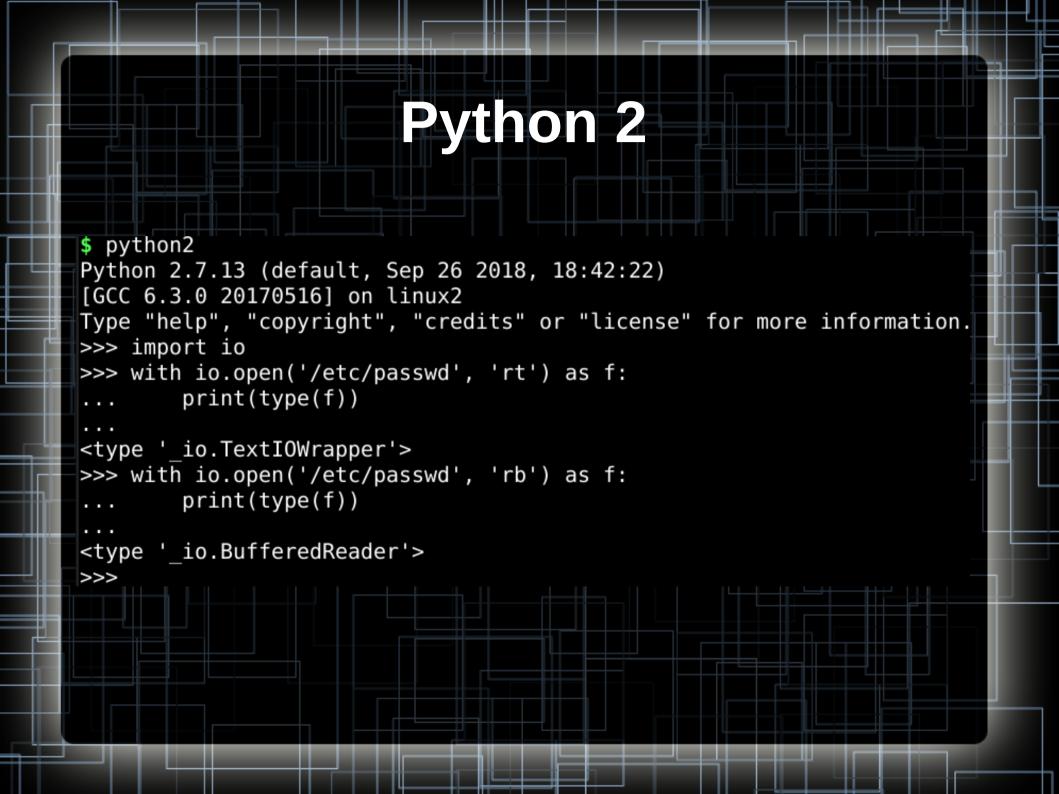
Ran with Python 2

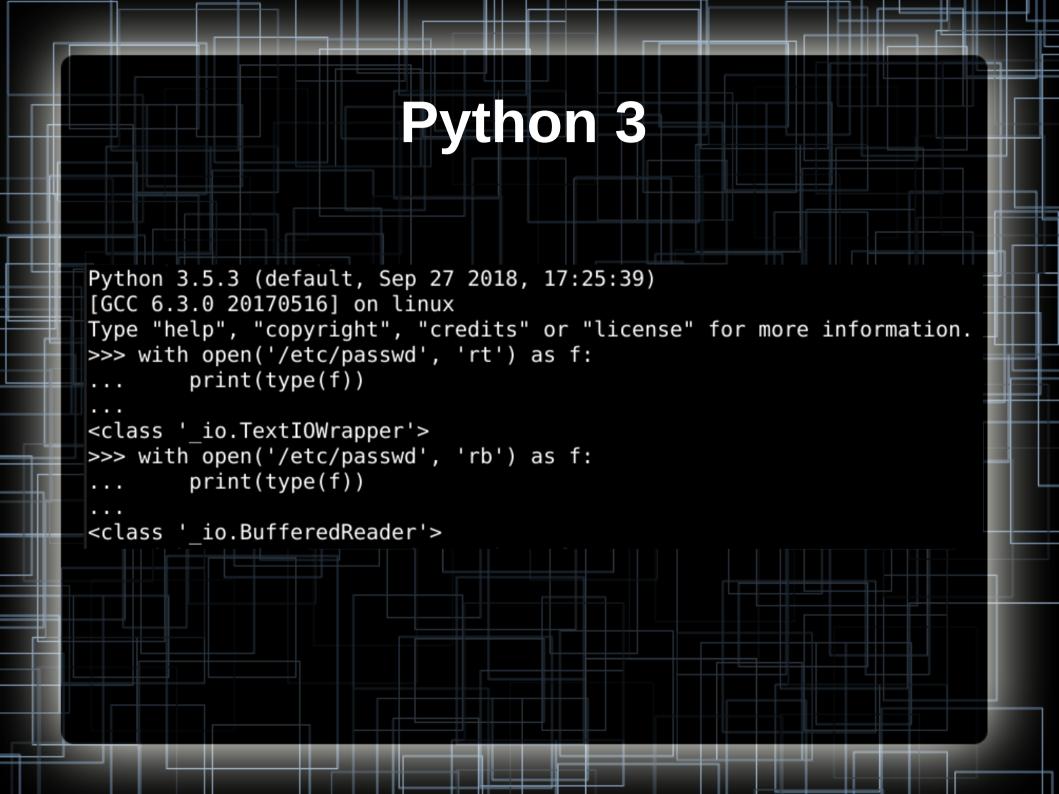
```
$ python2 ./argparse example universal.py
usage: argparse example universal.py [-h] file
Check if a file exist
positional arguments:
 file
             check if file is valid.
optional arguments:
  -h, --help show this help message and exit
$ python2 ./argparse example universal.py /etc/passwd
<type 'file'>
Seems to exist!
$ python2 ./argparse example universal.py /etc/passwwd
usage: argparse example universal.py [-h] file
argparse example universal.py: error: argument file: can't open '/etc/passwwd
: [Errno 2] No such file or directory: '/etc/passwwd'
```

Ran with Python 3

```
$ python3 ./argparse example universal.py
usage: argparse example universal.py [-h] file
Check if a file exist
positional arguments:
  file check if file is valid.
optional arguments:
  -h, --help show this help message and exit
$ python3 ./argparse example universal.py /etc/passwd
<class ' io.TextIOWrapper'>
Seems to exist!
$ python3 ./argparse example universal.py /etc/passwwd
usage: argparse example universal.py [-h] file
argparse example universal.py: error: argument file: can't open '/etc/passwwd'
: [Errno 2] No such file or directory: '/etc/passwwd'
```







Syntax Issues #6: Str Vs Bytes

- The bane of my existence at present...
- Iteration of strings returns: chr
- Iteration of bytes returns: int
- Ord() will error when getting ints
 - Not sure why it won't just return the int it got if 0 <= n <= 255</p>
- Base64.b64decode() behavior has changed as well.
 - Python 2: Str;Python 3: Bytes

Syntax Issues #6: Str Vs Bytes

- Bytes has some string functions, but misses some things like strip()
 - This require conversion to a string.
 - Be aware that you avoid unicode for this
 - Recommend 'latin-1' instead of 'UTF-8' if dealing with binary to avoid stupid encoding issues where it attempts to add unicode.
- Base64 encoded null padded data required:
 - Binary => String => call strip() => Binary

Library Issue #1: GTK2

- GTK2 has no bindings for python 3.
- There are bindings for GTK+.
 - Not a fan of the new structure of this personally.
- My plan was just port my GUI apps to use PyQT4 instead.
 - Works on python 2 and 3 just fine.

Library Issue #2: Pwntools

- Designed for 2.7
- Been Forked to 3 unofficially.
- Real devs are working on a fork to Python 3 (Dev3 Branch)
- Originally, I was keeping these scripts on 2.7 till the Dev3 Branch matures.
- The Dev3 Branch however seems to be working fine for Python 3 and Debian keeps python version libraries in two different folders.

Had Some Other Library Issues as Well

- But just needed to install their Python 3 counter parts:
 - Scapy
 - Jks
- Still running into some data conversion issues with ctypes however.
 - Seems like it has to do with the way strings and data are treated in 2.7 vs 3.

Wrap-up

- Overall, Python 2 and 3 support seems possible.
- Migration is a little slow, but do-able within the year.
- The new bytes vs str thing may make me consider other languages for binary analysis.
- Overall, Google is your friend here. There is plenty of talk on this topic in forums and blogs at this point in the game.

