THE POWER OF BEING UNDERSTOOD

RSM

PYTHON FOR PENTESTERS

About

Michael Benich, OSCP, OSWP

Supervisor

Penetration tester, former teacher

About

Spencer McIntyre, OSCP, OSEE

Manager

Spencer McIntyre is an open source security enthusiast and self-described Pythonista.

RSM

Learning Objectives

- By the end of this training, you should be able to:
  - Understand the structure of well-formed script module and package
  - Create and setup a development environment
  - Describe the core API, common functions, and third party packages
  - Identify opportunities for using Python to automate repetitive tasks

RSM

Introduction

Why should I care? I'm not a developer!

- Efficiency
  - More time for deeper analysis of "interesting" bugs
- Flexibility
  - Burp license expired? Kali update broke your tool? No problem!
- Knowledge
  - You might find a GraphQL endpoint that accepts valid Python ☺

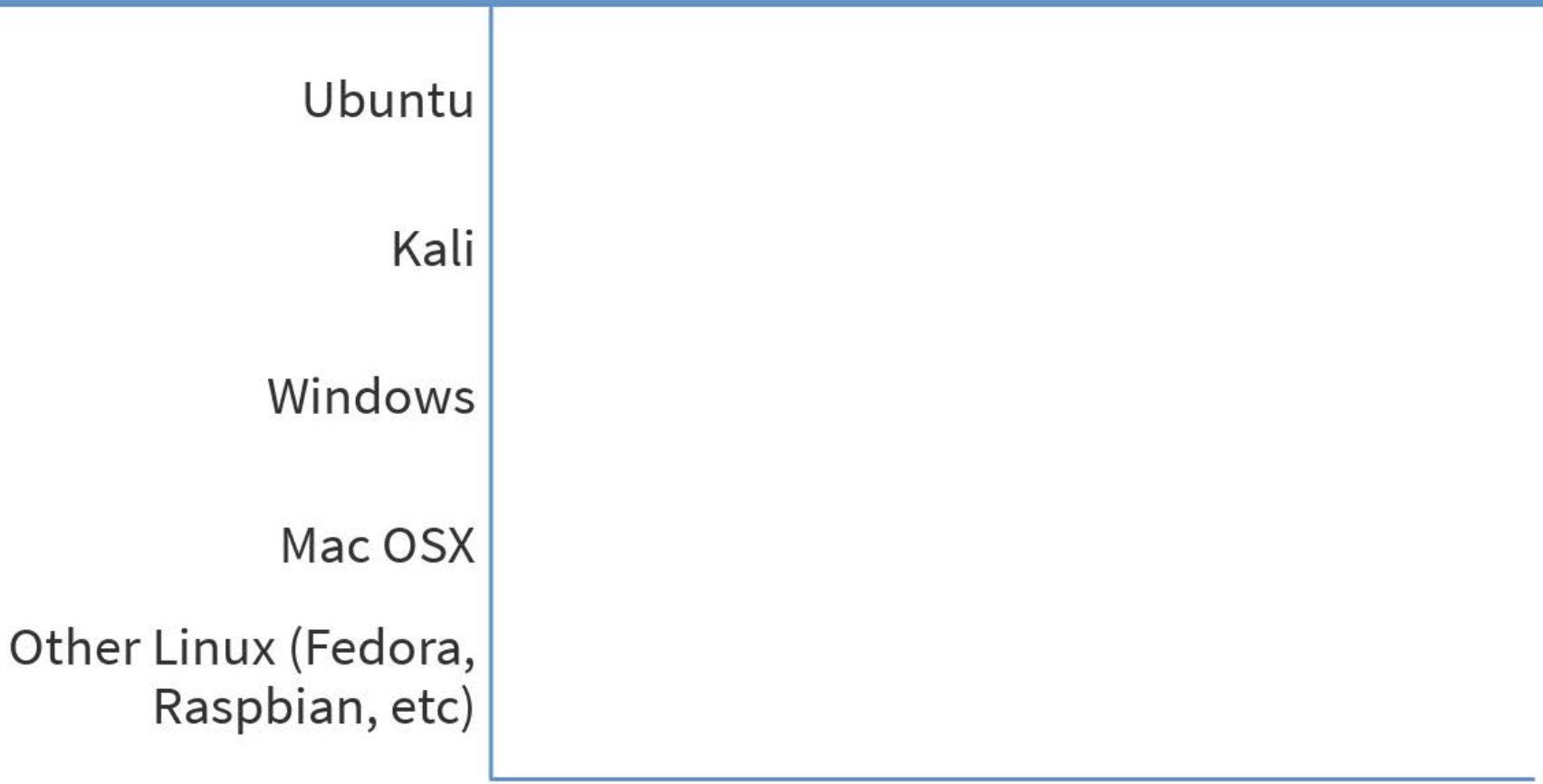
RSM

Lab Overview

- Hands on labs later in the course
- **Labs require a Python 3.6 or 3.7 environment!**
  - Start that setup now if you haven't already done so
- Labs will have difficulty "modes"
  - Select what's appropriate for your level
    - Easy mode – Update expressions
    - Normal mode – Write sections
    - Hard mode – DIY
- We'll provide the "reference implementation" later

RSM

What operating system are you most interested in running Python on?



SETTING UP A DEVELOPMENT ENVIRONMENT

Development Environment

Choosing an IDE:

- Atom
- PyCharm
- Sublime
- IDLE
- Spyder
- Jupyter
- Geany

Considerations

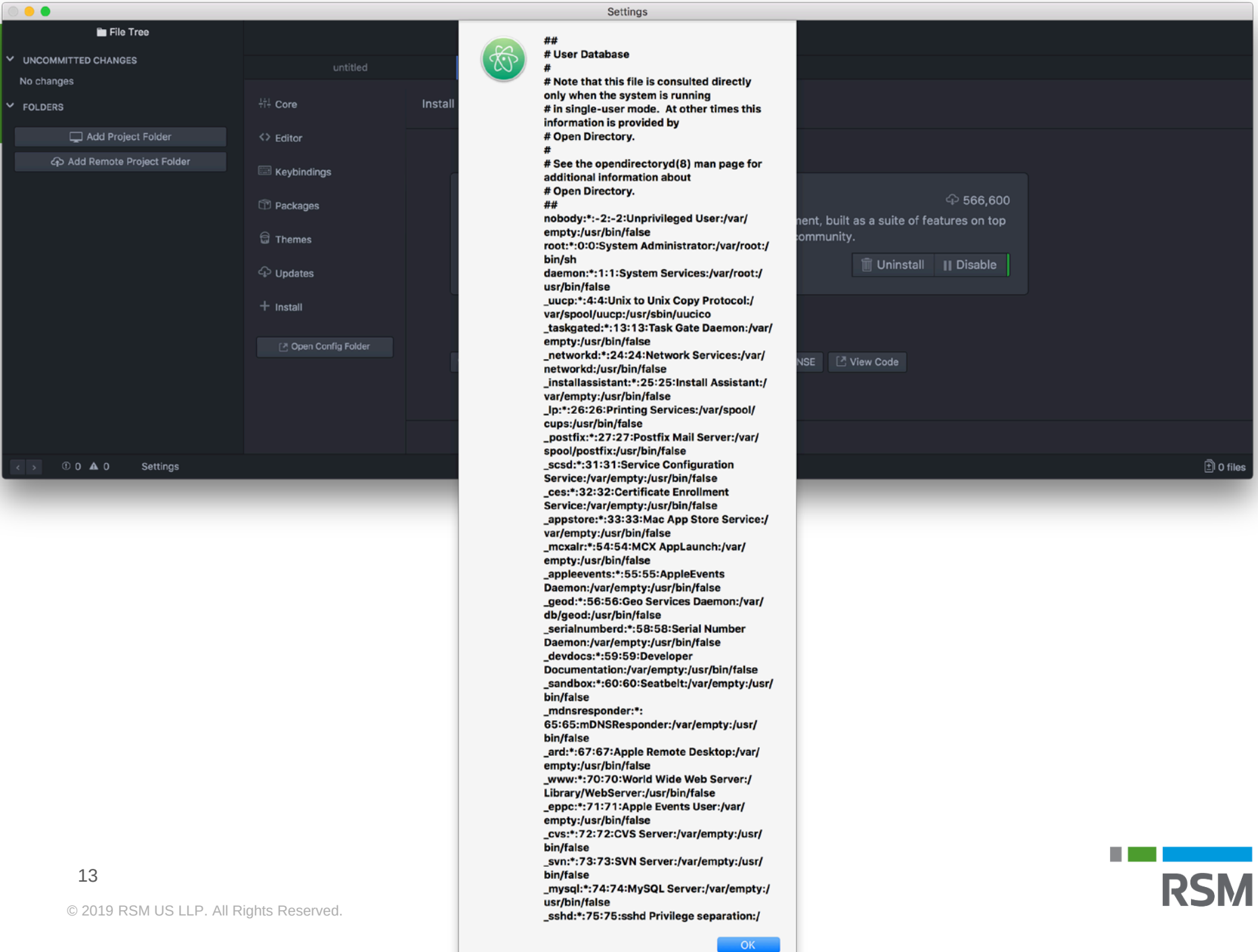
- Free or Paid?
  - If you are a student with a .edu email, you can get a free PyCharm license (jetbrains.com/student)
- How much RAM can you sacrifice?
  - Some of these Electron apps can eat up a lot of resources
- What are you trying to accomplish?
  - Quick, one-off script vs. maintaining an open source project

RSM

Enter Atom

- Good enough for this course
- Extensible with plugins / themes
- Downloadable from the “Releases” page on Github
  - https://github.com/atom/atom/releases/tag/v1.36
- Be careful with Electron apps ☺

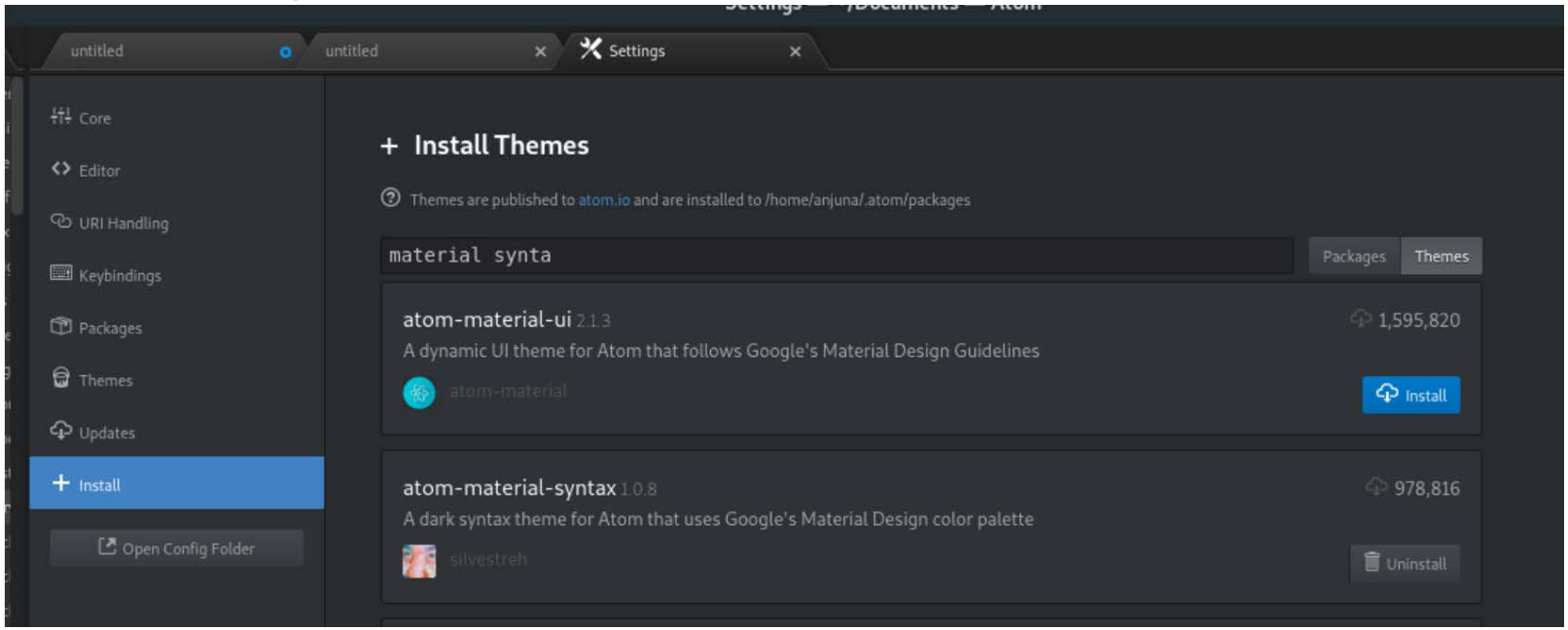
RSM



RSM

Install themes

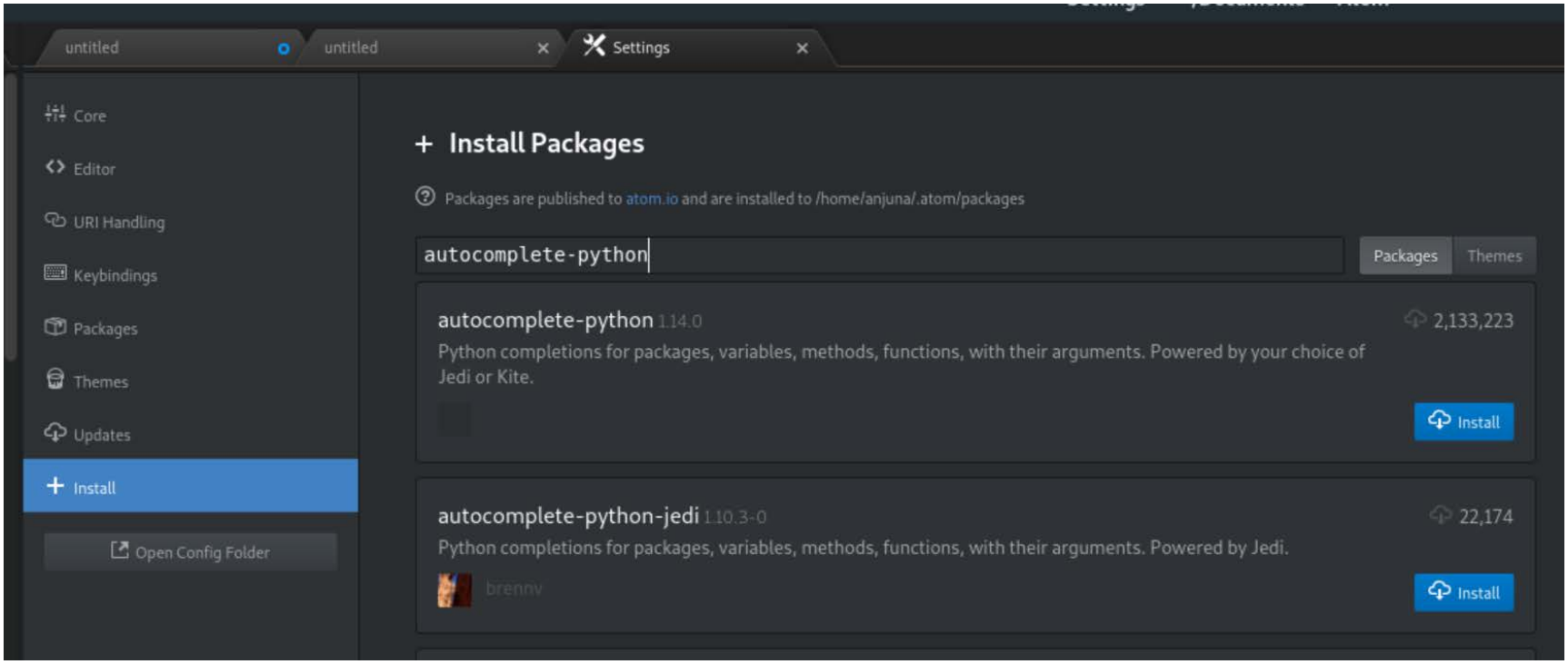
- Edit -> preferences
  - Themes – syntax themes
    - Up to preference, I prefer Material for projection because of the high contrast



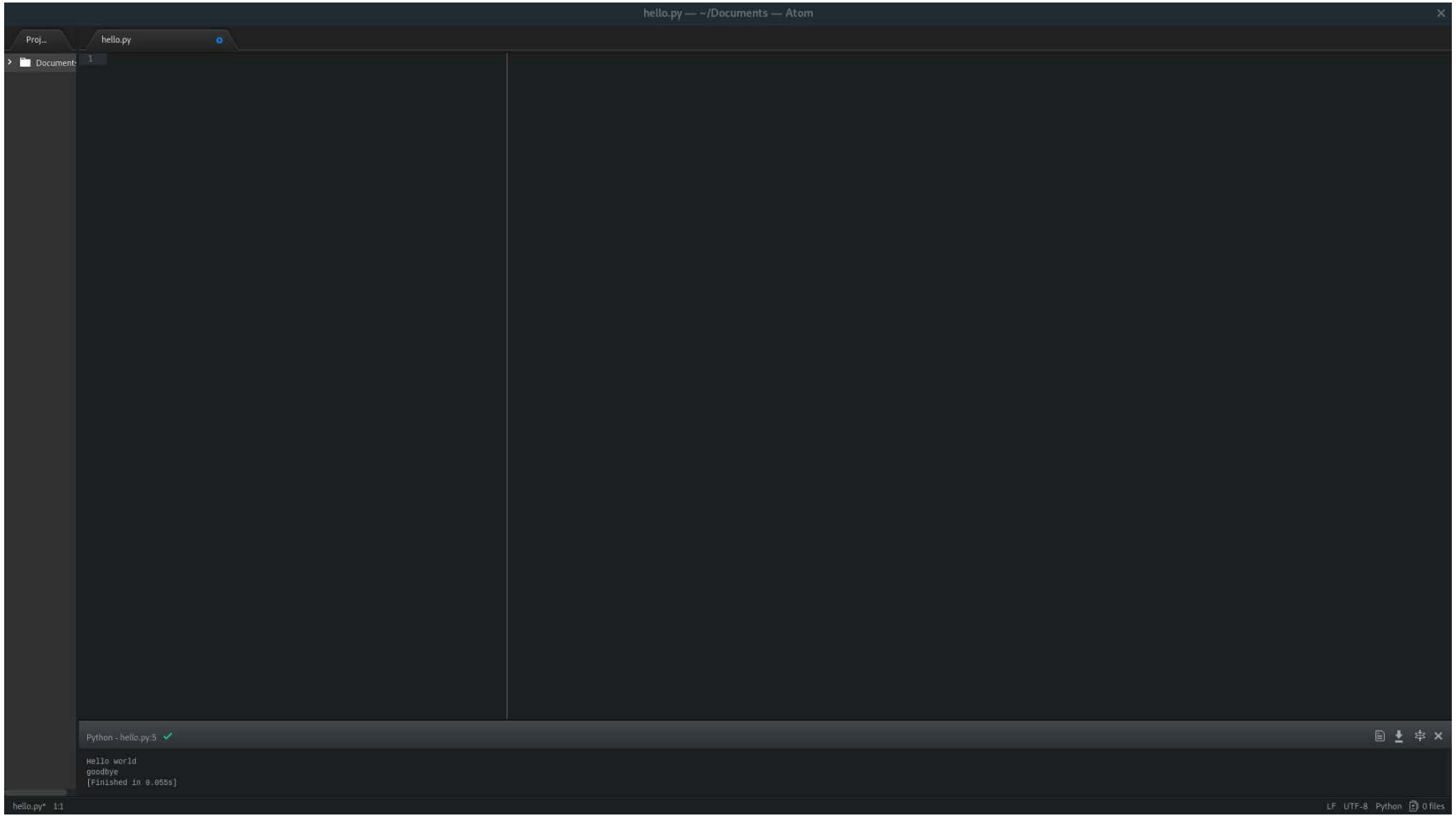
Install packages

Packages

– Python-autocomplete, autopep8, script, atom-ide-ui



Script: Shift-Control-B



Heavy-weight Alternative: PyCharm

- “Fully Featured” IDA (lots of features)
  - Best debugging, remote capabilities even
  - Remote file management (push / pull over SFTP)
  - Database integration
  - Powerful refactoring capabilities
    - Context-sensitive not just find / replace
  - Run / Debug configurations with templates
  - Pipenv / Venv integration
- Best used for large projects and complex tasks
  - More than ~3 source files

RSM

To pay or not to pay?

- PyCharm comes in a free and Paid Version
- Paid version includes additional features
  - Framework integrations like Flask, Django
  - Remote capabilities
    - File management
    - Debugging
  - Database manager
  - UML and SQLAlchemy diagram generation

RSM

crimson-forge [~/Repositories/crimson-forge] - .../crimson_forge/analysis.py [crimson-forge] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help HIDPI profiles

crimson-forge 〉 crimson_forge 〉 analysis.py x86 Reverse TCP Git:

Project

- crimson-forge ~/Repositories/crin
  - crimson_forge
  - data
    - stubs
      - amd64
      - macros.asm
    - templates
    - catalog.json
    - common-arguments.txt
    - crimson-forge-logo.png
  - docs
  - files
  - metasploit
  - tools
    - assembler.py
  - .gitignore
  - crimson-forge

analysis.py | assembler.py | block_api.asm

```
71      #     * stack: the address on the stack where the reference was pla
72      #     * referenced: the address that is referenced
73      Reference = collections.namedtuple('Reference', ('instruction', 'sta
74      ReferenceExpression = collections.namedtuple('ExpressionReference',
75      def __init__(self, blocks, copied=False):...
83
84      @classmethod
85      def _breakpoint(cls, state, event_type):...
93
94      def _breakpoint_expr(self, state):
95          expr = state.inspect.expr_result
96          if not expr.symbolic:
97              return
98          if expr in self.expressions:
99              return
100         if any(symbol.variables & expr.variables for symbol in self.ex
101             self.expressions[expr] = self.ReferenceExpression(expr, Non
102             logger.info('New tainted expression identified')
103
104     def _breakpoint_mem_read(self, state):...
124
125     def __addr_is_on_stack(self, state, stack_addr, address):...
136
137     def __breakpoint_mem_write(self, state, stack_addr, ip_addr):...
157
```

SelfReferenceTracker 〉 _breakpoint_expr()

6: TODO 9: Version Control Terminal Python Console Event Log

95:41 LF ÷ UTF-8 Tab ÷ Git: master ÷

Git

- Version control system
- Advantages:
  - Saving and tracking incremental changes
  - Collaborating with others
  - “Topic” branches to test new features (eg, master vs. dev vs. experimental)
- Can easily publish to a site such as Github, Gitlab, Bitbucket, Keybase… etc

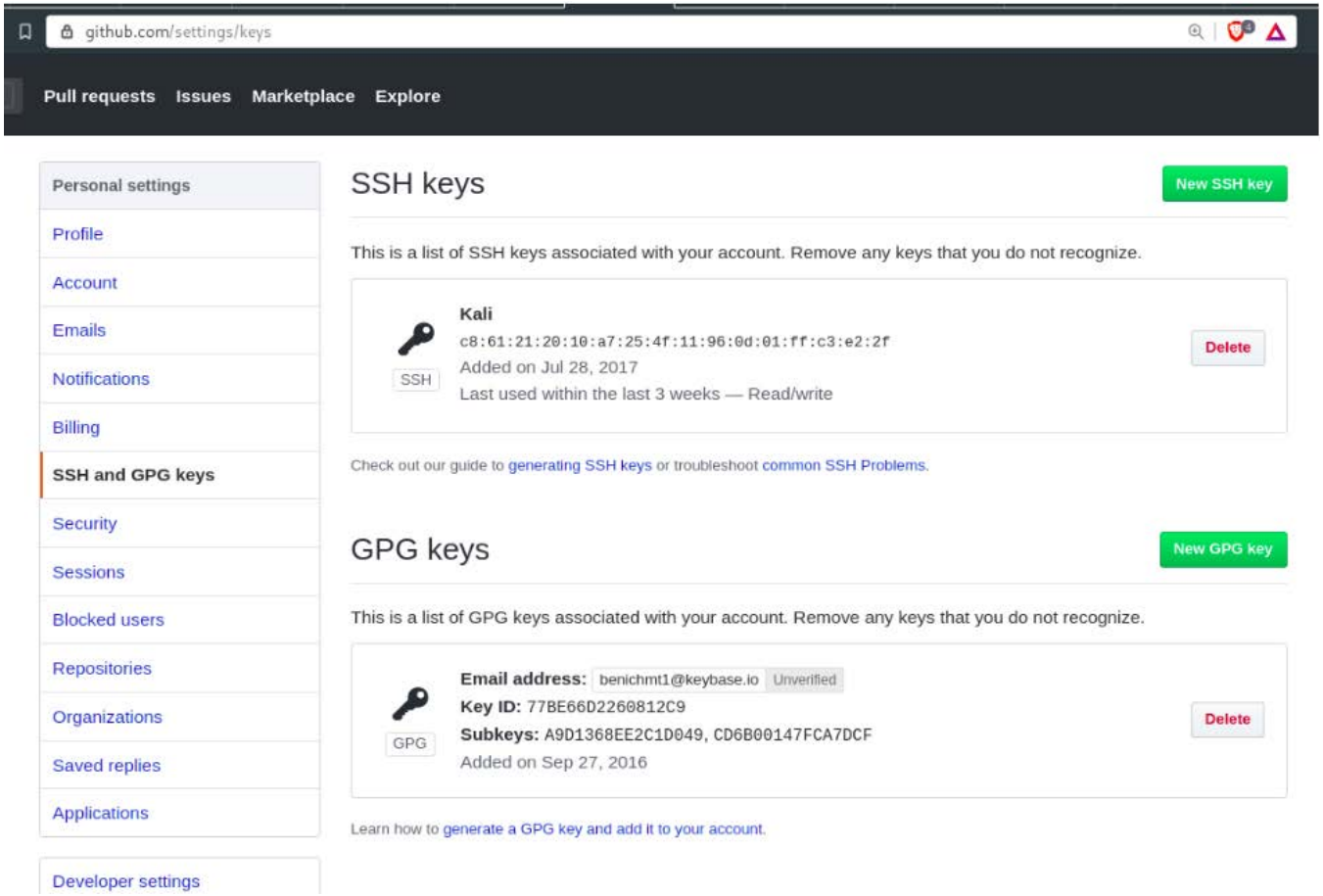
RSM

Git basics

- New project?
  - `git init`
    - Creates a .git file
- Working off someone elses?
  - `git clone https://github.com/securestate/king-phisher.git`
  - `git clone –b dev <url>`
- Made your changes and you want to record them?
  - `git commit –m “Updated helper function”`
- Forgot the last thing you did?
  - `git log`
- Ready to publish your changes?
  - `git push`

RSM

Git keys

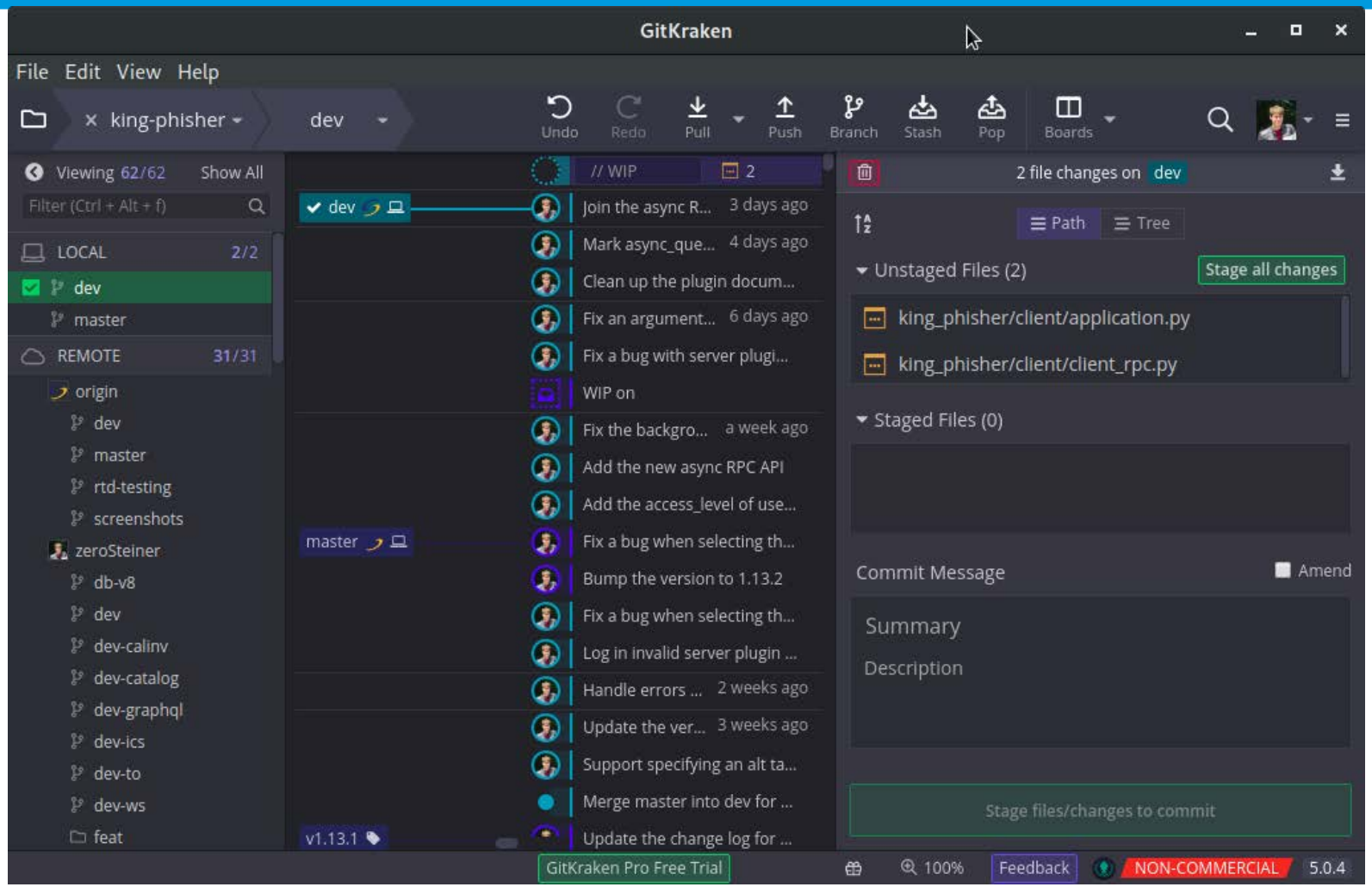


Git GUIs

- GUIs are ideal for complex Git tasks
  - Merging / Rebasing
- Searching through commit logs
- Cherry-picking commits

RSM

GitKraken



Pip

- Pip – python package manager
  - `pip / pip2 / pip3 install <package name>`
- Previously if you wanted to get going with a project, you would `git clone` it, and then `pip install –r requirements.txt`
- This usually solved most of the errors that stem from not recognizing an import
- But what about when requirements conflict, or are not backwards compatible?

RSM

Enter PIPENV

- Allows developers to replicate the same environment
  - `Why does it work on YOUR machine but not on mine?`
- Can replicate the exact same environment, PER PROJECT (eg, we have all the same requirements, and they are all a specific version)
- `pip install pipenv`

RSM

PipEnv in 60 seconds

- `Pipenv shell`
  - Creates an environment
- `pipenv install dependency==1.0.0`
  - Installs dependency version
- Can replicate the exact same environment, PER PROJECT (e.g., we have all the same requirements, and they are all a specific version)
- `pipenv graph`
  - A list of the dependencies you have
- `pipenv lock`
  - Updates your lock file when you are ready to go

RSM

PYTHON BASICS

Python Variables

- Python is often recommended to beginners for many reasons, one of which is dynamic variable types
- Python tries to “autodetect” the variable type for you (compared to C++/Java where you have to define types)
  - `C++: int i = 1; (i is an integer type)`
  - `Python: i = 1 (i is assumed to be an integer)`

RSM

Python Types

- X = 1
  - Int (integer)
- X = 1.0
  - Float (calculations with precision)
- X = “one”
  - String (character or characters)
- X = [1,2,3]
  - List
- X = (1,2,3)
  - Tuple
- {“x”:1}
  - Dict (dictionary)

RSM

Mutable / Immutable Objects

Mutable	Immutable
list	str
dict	int
set	float
Can change its contents	Cannot change its contents

RSM

Mutable / Immutable Objects

```
print("-------------- Immutable (int)-------------“)
x = 1
y = x
print("The object id of x is: {}".format(id(x)))
print("The object id of y is: {}".format(id(y)))
y = y+1
print("The value of y is {}".format(y))
print("The value of x is {}".format(x))
print("The object id of x is: {}".format(id(x)))
print("The object id of y is: {}".format(id(y)))
```

Mutable / Immutable Objects

```
print("-------------- Mutable -------------“)
a = [1,2,3]
b = a
print("The object id of a is: {}".format(id(a)))
print("The object id of b is: {}".format(id(b)))
a.append(4)
print("Do these have the same identity? {}".format(a is b))
print("The object id of a is: {}".format(id(a)))
print("The object id of b is: {}".format(id(b)))
print("The value of a is: {}".format(a))
print("The value of b is: {}".format(b))
```

Warning

```
def bad_function(thing, items=[0, 1, 2]):
    items.append(thing.favorite_item)
    return process_items(items)
```

- Don’t use mutable types in function signatures
  - The default items instance will persist between function calls

RSM

String Format Operations

```
company = "RSM"

years = 3

dollars = 100000.0


print("I have worked for %s for %d years and earned %.2f
dollars" % (company, years, dollars))


print("My friend also works for {0} for {1} years and earned
{2:.2f} dollars".format("Charlie's Chocolate Factory", 5,
dollars+34000))
```

Functions

```
print("----- Simple Function ------“)

def reversefunction(company):
    bwstring = ''.join(reversed(company))
    return bwstring
myco = "RSM"
print("{} backwards is {}".format(myco,reversefunction(myco)))
```

Using *args

```
print("--- Using *args ----“)
def moneyfunction(*pricelist):
    total = 0
    for n in pricelist:
        total = total + n
    print("The total cost is ${} ".format(total))
pricesheet = (24,58)
moneyfunction(*pricesheet)

pricesheet2 = (17,37,21,442)
moneyfunction(*pricesheet2)
```

**kwargs

```
print("--- Using **kwargs ----“)
def employeedata(**data):
    for key, value in data.items():
        print("{}:{}".format(key,value))

employeedata(Name="John", Age="22")
employeedata(Name="Sandy", Location="Ohio")
```

**kwargs usage

```
# Unconditionally set consultant keys
def consultantdata(**data):
        data['Type'] = 'Consultant'
        data['Utilization'] = 1.0
        return employeedata(**data)

# Set default consultant keys
def consultantdata(**data):
        data['Type'] = data.get('Type', 'Consultant')
        data['Utilization'] = data.get('Utilization', 1.0)
        return employeedata(**data)
```


RSM

Working with Files

- Files can be opened with various modes
  - r (read only)
  - w (write only)
  - r+ (read and write)
  - a (append)
  - b (binary)
- As a simple example, you can read a file from the same directory as the script is in
- f = open("input.txt","r")
- g = open("output.txt","w")

RSM

Reading and Writing Text Files

- Once you open a text file, you can process it in a number of different ways
  - **read** – read all into one string
    - Good for shorter files
  - **readline** – read individual lines
  - **readlines** – read all lines and return as a list
- **Warning:** Opening a file in write mode (without “r”) will overwrite the contents
  - Will also create a new, empty file

RSM

Reading and Writing Text Files

```
f = open("input.txt","r")
g = open("output.txt","w")
sentencesfromfile = f.readlines()
for line in sentencesfromfile:
        print(line)
print("The first line is:")
print(sentencesfromfile[0])
g.write(sentencesfromfile[0])
f.close()
g.close()
```


RSM

Exception Handling

- Try to keep the body of the try: block as small as possible
- Do not catch all Exceptions without logging useful information
  - If you need to use blanket exception handling log a stack trace
    - `traceback.print_exc` will explicitly print the stack trace
    - Logging `exc_info=True` will log the stack trace (only useful if logging is configured

RSM

Exception Handling: Bad Example

```
• try:
    user = ‘alice’
    data = some_faulty_method(user)
    data += some_other_method(user)
• except Exception:
    print(‘something went wrong’)
```

Exception Handling: Good Example

```
• user = 'alice'
• try:
    some_faulty_method(user)
• except TypeError:
    traceback.print_exc()
• else:
    data += some_other_method(user)
• finally:
    logout(user)
```

What is the output of the following code?

main.py saved

```
def myfunction():
    try:
        print("a")
    except TypeError:
        print ("b")
    finally:
        print("c")
myfunction()
```

a

a b

a b c

a c

b

c

PYTHON PACKAGES

Python Packages and Modules

- Python can be extremely useful and timesaving through the use of packages – don't reinvent the wheel!
- Packages can be installed via pip.
  - Pip install –r requirements.txt
  - Pip install requests
- One of the most basic troubleshooting skills – did you install all of the required packages first?
- For this course, we will not be creating our own packages, but we will show the usefulness of working with others'.
- Packages are collections of modules – a module is a singular import

RSM

Python Modules

```
import os
os.listdir()
my_directory = os.getcwd()
print(my_directory)
```

Python Modules

```
import os, re
if os.path.isfile("/etc/vsftpd/vsftpd.conf")
        fileopen = open("/etc/vsftpd/vsftpd.conf","r")
        data = fileopen.read()
        match = re.search("anonymous_enable=YES",data)
        if match:
                print("[!] Insecure configuration! Anonymous FTP enabled!")
```


RSM

Command Line Arguments

Arguments are the placeholders

python3 file.py **input.txt**

```
import sys
print(“Arguments:",len(sys.argv)-1)
print("Your first argument is:",(sys.argv[0])
print("Your second argument is:",(sys.argv[1])
```

Command Line Arguments

A more robust way of dealing with user input is through **argparse**

```
import argparse

parser = argparse.ArgumentParser()

parser.add_argument("-V","-v","--version",help="show the version",
action='store_true')

parser.add_argument("-F","-f","--face",help="show the weird face
emoji",action='store_true')

parser.add_argument("-P","-p","--port",help="change the port")

args = parser.parse_args()

if args.version:

    print("Web Server 0.0/0")

if args.face:

    print("0_______o")

if args.port:

    print(args.port)
```

Debugging Python

- What is debugging?
- Allows you to inspect the state of things as they're running.
  - Check values
  - Trace logic

RSM

Options

- CLI:
  - pdb – Integrated Python debugger, pretty basic
  - ipdb – pdb, but with IPython-like capabilities
  - rpdb – Remote PDB, a network-capable variant of pdb
- GUI:
  - PyCharm – Heavy weight IDE for Python (authors choice)

RSM

pdb / ipdb / rpdb

- All command line based, easy to use when running a script as root
- Only pdb is included in Python, use it in restricted environments
- Use rpdb when connecting to stdin / stdout isn't possible
  - Where Python forks into the background, injects in another process etc.
  - Opens a socket, connect in with netcat

RSM

Starting The Debugger

- Easiest is to set a “breakpoint”
  - The script will pause at this point for inspection
- `pdb.set_trace()`
  - rpdb, ipdb all have set_trace functions
- New `breakpoint()` built in function (Python 3.7+)
- Commands are intermixed with expressions
  - Prefix an expression with ! if it overlaps

RSM

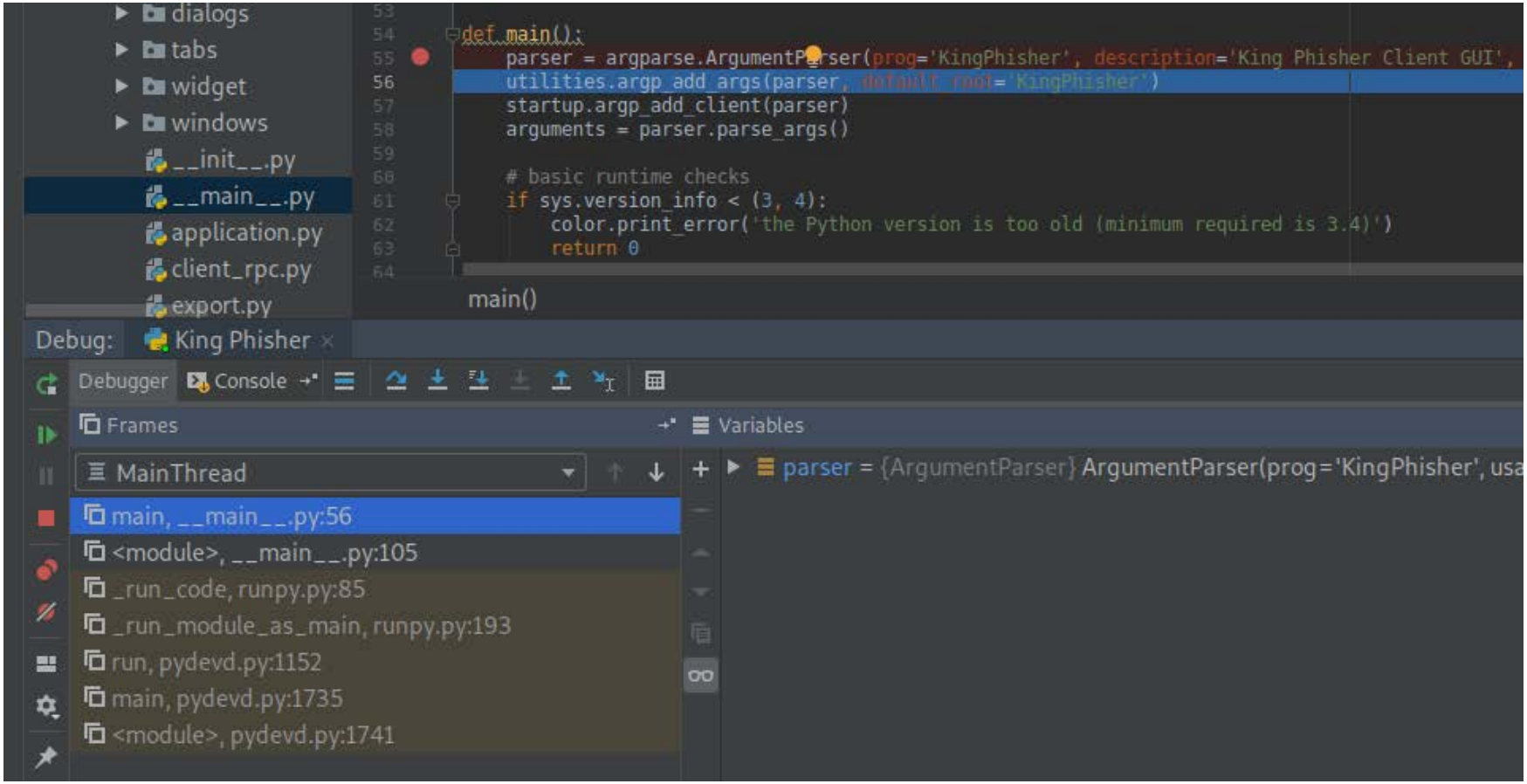
Basic Debugging Commands

- next – Run the next line of code, **stepping over functions** and then pause
- step – Run the next line of code, **stepping into functions** and then pause
- continue – Resume normal execution
- where – Print a stack trace

PyCharm Debugger

- Editor's Choice
- Community (Free) version available, includes the debugger
- Has some advanced features
  - Pause all threads vs just one
  - Conditional breakpoints
  - Watchpoints
  - Stores breakpoints across sessions

RSM



Stack Trace

- Each “frame” of the stack trace takes two lines
- First line is file and line number
- Second (indented) is the expression
- Lower frames are more recent
  - Last line is most often an exception
- Variable values are not included

RSM

Stack Trace Example

```
Traceback (most recent call last):
  File "/home/saurus/king-phisher/king_phisher/client/application.py", line 488, in do_server_connected
    campaign = self.get_graphql_campaign() if campaign_id else None
  File "/home/saurus/king-phisher/king_phisher/client/application.py", line 371, in get_graphql_campaign
    campaign = self.rpc.graphql_find_file('get_campaign.graphql', id=campaign_id)
  File "/home/saurus/king-phisher/king_phisher/client/client_rpc.py", line 241, in graphql_find_file
    return self.graphql_file(path, query_vars=query_vars)
  File "/home/saurus/king-phisher/king_phisher/client/client_rpc.py", line 224, in graphql_file
    return self.graphql(query, query_vars=query_vars)
  File "/home/saurus/king-phisher/king_phisher/client/client_rpc.py", line 205, in graphql
    query_vars=query_vars
king_phisher.errors.KingPhisherGraphQLQueryError
```

Start Debugging

- Set a breakpoint on the line before the exception
  - In our example this would be client_rpc.py:204
- Inspect the values
  - If the code works “sometimes”, a value is probably not what you expect
  - If the code has never worked (you’re writing it) the usage is probably incorrect
    - Look up the documentation for what you’re using

RSM

Common Exceptions

- TypeError – A value of an incompatible type was used e.g. using a string when an integer was expected
- KeyError – A key doesn't exist in a dictionary
  - `my_data['this does not exist']`
- AttributeError – A requested attribute doesn't exist
  - my_data.this_does_not_exist

RSM

SCRIPT STRUCTURE

Script Structure

- Top of the file will be the comment splat
  - This usually includes the license information (BSD, GPL, MIT, etc.)
- Import statements
  - Pro-tip: group these by origin, for example native Python, current package, and third-party

RSM

Script Structure

- Any values including constants
- Functions, classes, exceptions
- Finally if the file is a script utilize:
- `if __name__ == '__main__':`

  `main()`
- This allows the file to be executable and importable
  - The entry point is the `main()` function, so ensure it's defined

Script Structure

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#
#  protocon
#
#  Redistribution and use in source and binary forms, with or without
#  modification, are permitted provided that the following conditions are
#  met:
#
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above
#    copyright notice, this list of conditions and the following disclaimer
#    in the documentation and/or other materials provided with the
#    distribution.
#  * Neither the name of the project nor the names of its
#    contributors may be used to endorse or promote products derived from
#    this software without specific prior written permission.
#
#  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
#  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
#  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
```

```python
import argparse
import functools
import os
import sys

get_path = functools.partial(os.path.join, os.path.abspath(os.path.dirname(__file__)))
sys.path.append(get_path('lib'))

import protocon
```

```python
EPILOG = """\
target_url examples:
  null:
  serial:///dev/ttyUSB0?baudrate=9600&bytesize=8&parity=N&stopbits=1
  tcp://1.2.3.4:123
  tcp4://0.0.0.0:123/?type=server
  tcp6://[fe80::800:27ff:fe00:10]:4444/?ip6-scope-id=eth0
  udp://1.2.3.4:123
  udp4://1.2.3.4:123/?size=8192
"""
```

```python
def main():
        parser = argparse.ArgumentParser(description='protocon', conflict_handler='resolve', formatter_class=argpars
        parser.add_argument('-q', '--quiet', action='store_true', default=False, help='initialize quiet to True')
        parser.add_argument('-v', '--version', action='version', version='%(prog)s Version: ' + protocon.__version__
        parser.add_argument('target_url', help='the connection URL')
        parser.add_argument('scripts', metavar='script', nargs='*', help='the script to execute')
        parser.epilog = EPILOG
        arguments = parser.parse_args()

        try:
                engine = protocon.Engine.from_url(arguments.target_url)
        except protocon.ProtoconDriverError as error:
                protocon.print_error('Driver error: ' + error.message)
        else:
                engine.entry(arguments.scripts)
                engine.connection.close()
        return 0

if __name__ == '__main__':
        sys.exit(main())
```

- https://github.com/zeroSteiner/protocon/blob/master/protocon

Script vs Module vs Package

- Scripts and Modules are single files
  - Modules and packages are sometimes generically called “libraries”
- A module can also be a script
  - Scripts are executable from the command line
  - Modules are `import`-able
- A package is a directory with one or more modules
  - Must have a __init__.py file (can be empty)
  - Package can have sub-packages

RSM

HTTP REQUESTS

Making HTTP Requests

- The painful way (using the builtin modules)
  - urllib.request
- The easier way: use requests
- Following example will show the same POST request
  - With a proxy
  - With parameters
  - Ignore self-signed SSL certificates

RSM

Using urllib.request

- `params = urllib.parse.urlencode({'id': 123}).encode('utf-8')`
- `context = ssl.create_default_context()`
- `context.check_hostname = False`
- `context.verify_mode = ssl.CERT_NONE`
- `req = urllib.request.Request(url, params, context=context)`
- `req.set_proxy('localhost:8080', 'http')`
- `resp = urllib.request.urlopen(req)`
- `data = resp.read()`

RSM

Using requests

- `requests.post(url, data={'id': 123}, verify=False)`

- But wait, where's the proxy?
  - It's in the `HTTP_PROXY` environment variable… where it belongs

Lab Time – Objective

- Build an HTTP bruteforcer / crawler
- Start with a URL and a wordlist
  - Check the URL for subpaths from the wordlist
  - For each page that is found, parse it for links
- Helps with enumeration for web applications
  - Tweak it targeted tasks (ie, looking for specific file types)
- Script is incomplete, complete the lines listed with todo

RSM

Lab Time – Recap

- Anyone have any questions?
- Did anyone add any features or otherwise improve the script?

REGULAR EXPRESSIONS PRIMER

What are Regular Expressions

- Also abbreviated as regex
- A patter for matching text
- Not specific to Python
  - Any substantial language includes support
    - Perl, Ruby, C, Javascript, Powershell
  - Also command line tools like grep
  - Some have slight syntatic differences but are for the most part the same

RSM

Regular Expressions

- Given the string: “Hello, my name is Spencer”
- To match regardless of name
  - “my name is \w+”
- The “my name is” portion is static
- The “\w+” portion is *one or more instances of the \w set*
  - \w is a shortcut for [a-zA-Z0-9_] and is functionally equivalent
  - The same regex could be “my name is [a-zA-Z0-0_]+”

RSM

Character Sets

- Important part of regexs, provides the “fuzzy” matching
- Sets are enclosed in square brackets and can include ranges
  - All numbers: “[0-9]”
  - Only vowels: “[aeiou]”
  - Anything but letters: “[^a-z]”
    - The leading ^ implies “anything but this set”, this example would include numbers, whitespace and special characters

Common Character Sets

- \d – Decimal digits
- \s – Whitespace (spaces, tabs, newline etc.)
- \w – Word characters (alphanumeric and underscore)
- Any of these can be inverted by capitalizing their respective letter

RSM

Repeating Characters

- \w matches “a” but to match “aa” use a range
- \w* – *zero or more instances of \w*
- \w+ – *one or more instances of \w*
- \w{3} – *exactly three instances of \w*
- \w{3,6} – *between three and six instances of \w*

Groups

- Useful for extracting text
- Enclose the pattern in parenthesis
  - “my name is (\w+)” will extract the name
- Python supports named groups, useful for large regexs
  - “my name is (?P<name>\w+)”

RSM

Logical OR

- Use | (the pipe character)
- a|b – matches either a or b
- Example: “(hello|goodbye)\sSpencer”
  - Matches both:
    - “hello Spencer”
    - “goodbye Spencer”

Writing Regexs

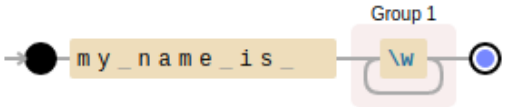
- Online tools are helpful
  - https://www.debuggex.com
  - https://regexcrossword.com
- Use Python's "raw" strings (r prefix)
  - r"some(regex)"
  - Makes escaping easier

RSM

Debuggex.com

Untitled Regex No description

Embed on StackOverflow



Python

View Cheatsheet

Flags

```
1  my name is (\w+)
```

Result: **Does not match** starting at the black triangle slider

```
1  Hello, my name is Spencer
```

START

POSITION

Regex Module Primer

- Python provides the builtin re module
- Pro-tip: `re.compile` isn't necessary
  - All the main methods are cached automatically
- Regexs are great for parsing and searching text-based file, not binary

RSM

Main Regex Methods

- `match()` vs `search()`
  - `match()` requires that the start of the string matches the regex
- Example:

```
re.match(r'Liddle', 'Alice Liddle') # =>
  failure

re.search(r'Liddle', 'Alice Liddle') # =>
  success
```

Regex Flags

- Flags modify the behavior of regexs
- Two most useful:
  - IGNORECASE / I
    - Match regexs while ignoring the case of alphabetic characters
  - MULTILINE / M
    - Changes ^ / $ to match the beginning and end of lines instead of the start and end of the string
- Combine flags with the bitwise OR
  - `flags=re.IGNORECASE | re.MULTILINE`

RSM

Flag Example

- # case sensitive version does not match
- re.match(r’alice’, ‘Alice’) # => no match
- # case insensitive version does match
- re.match(r’alice’, ‘Alice’, flags=re.IGNORECASE)

Extracting Information

- Use Python's named groups
  - Allows the regex to be updated without breaking the group index
  - (?P<name>…)
- Example:

```
match = re.match(
    r'^\s+password:\s+(?P<password>\S+)$',
    '  password: supersecret'
)
match.group('password') # => 'super secret'
```

Additional Regex Methods

- `sub()`
  - Replace parts of a string with another string or a callback function
- `findall()`
  - Find all matches of a regex within a string
  - Don’t forget the `MULTILINE` flag

Regex Crosswords

Tutorial The OR symbol

A|B

A|Z A

Validate

Next puzzle coming up in 2 seconds

« 1 2 3 4 5 6 7 8 9 »

Home

Lab: Regex Crossword Puzzles

- https://regexcrossword.com/
- We'll spend 15 minutes here
- Solve as many as you can
- If you know regexs, start with the higher levels

RSM

PARAMIKO PRIMER

Paramiko Overview

- Fully Python implementation of SSH
  - Includes a server and a client
  - Server implementation is much newer and has less documentation and examples
- Works best with OpenSSH, but can be used for network devices like Cisco IOS, etc.
- Supports running commands, SFTP, TCP forwarding, etc.

RSM

Paramiko Connections

- Initialize the SSHClient instance (a lower-level Transport class is available as well)
- Set the missing host key policy
  - Technically optional, but necessary if connecting to new devices
- Connect and authenticate to the server

RSM

Paramiko Connection Example

```
• # initialize the client
• ssh_client = paramiko.SSHClient()
• # define and add the policy
• policy = paramiko.AutoAddPolicy()
• ssh_client.set_missing_host_key_policy(policy)
• # connect and authenticate
• ssh.connect(target, port, username=username,

  password=password)
```

Executing Commands

- Use the exec_command method
  - Returns stdin, stdout, stderr streams
  - Being streams, these must be read
- Specify `get_pty=True`
  - Changes the execution behavior to more closely resemble running the command through a shell

RSM

SSH Command Example

```
# this is a long line
stdin, stdout, stderr = \
ssh_client.exec_command('env', get_pty=with_pty)
# read from the stdout stream
data = stdout.read()
# decode the stream to a string
data = data.decode('utf-8')
```

Paramiko SFTP

- From a connected client, open an SFTP channel
  - This returns a new SFTP object
- Provides basic methods for
  - Opening files
  - Listing directories
  - Reading and changing permissions

RSM

Paramiko SFTP Example

- `sftp = ssh_client.open_sftp()`
- `file_stat = sftp.stat('path/to/some/file')`
- `# use the stat module flags and file_stat.st_mode`
- `file_handle = sftp.open('path/to/some/file', 'r')`
- `# paramiko ignores the 'r' flag, always uses bytes`
- `data = file_handle.read()`
- `data = data.decode('utf-8')`

RSM

Lab Time - Objective

- Write a script that will SSH into a Linux server, read a file and extract data using a regex
- Multiple file paths, and regexs are provided
- Again the script is incomplete

Lab Time – Recap

- Anyone have any questions?
- Did anyone add any features or otherwise improve the script?

Python in your toolset

Burp Extensions

- Requires the Jython .JAR file to run Python

```
from burp import IBurpExtender
```

https://laconicwolf.com/2018/04/13/burp-extension-python-tutorial/

Powershell Empire

- Powershell is in the name, but modules are written in Python
- https://github.com/EmpireProject/Empire

Immunity Debugger

- Python API
- https://www.corelan.be/index.php/2010/01/26/starting-to-write-immunity-debugger-pycommands-my-cheatsheet/

RSM

Python is great!

- Mahatma Gandhi

THANK YOU FOR YOUR TIME AND ATTENTION

QUESTIONS AND ANSWERS

TEST YOUR KNOWLEDGE

RSM US LLP

Address
City
Phone

+1 800 274 3978
rsmus.com

RSM