# Lecture 3: Python for Security

BKACAD's Security Training

# Table of Content

# Table of Content

# Python & pyGames

## Python in Terminal

```
root@kali:~# python
Python 2.7.3 (default, Jan  2 2013, 16:53:07)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

# Python & pyGames

## Script Structure

```
1.  #! /usr/bin/python
2.  # User can comment a single line with a pound sign
3.  '''''
4.  The first string in the program is the DocString and is used by
5.  help function to describle the program
6.  '''
7.  import sys
8.  def main():
9.      'A "Docstring" for the main function here'
10.     print 'You passed the argument: ' + sys.argv[1]
11.
12. if __name__ == '__main__':
13.     main()
```

# Python & pyGames

## Introducing pyGames

- pyGames is a CTF for lecture 03
- Included about 20-30 challenges about basics in using Python
- 20 harder challenges are real-world scenarios in which coding skills are required
- Download: gitclone git@github.com:bkacadsec/sec.git

# Python & pyGames

pyGames – 1st terminal

```
root@kali:~# cd Lecture3/
root@kali:~/Lecture3# ls
pyGames.py    pygamesserver.py  questions.data
pyGames.pyc   python Basic.pdf  readme.txt
root@kali:~/Lecture3# python pygamesserver.py
```

# Python & pyGames

## pyGames – 2nd terminal

```
root@kali:~# cd Lecture3/

root@kali:~/Lecture3# python

Python 2.7.3 (default, Jan  2 2013, 16:53:07)

[GCC 4.7.2] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> import pyGames

>>> game = pyGames.game()

>>> help(game)
```

# Python & pyGames

Playing pyGames – 2nd terminal

```
>>> game.question(0)
'Simply return the data as the answer. '
>>> game.data(0)
'SUBMITME'
>>> game.answer(0, game.data(0))
'Correct!'
>>> game.score()
'You have 1 point(s), completed questions: 0 '
>>>
```

# Python & pyGames

## Exercises

1. pyGames question #0
2. pyGames question #1

# Table of Content

# Language Basics

Variable

Variable name are case-sensitive labels for object in memory

```
>>> x = 1
>>> y = 1
>>> z = "Hello"
>>> id(x)
137941168
>>> id(y)
137941168
>>>
```

# Language Basics

Variable

Variable types, types are also assigned automatically

```
>>> type(x)
<type 'int'>
>>> type(z)
<type 'str'>
>>>
```

# Language Basics

Variable

Reassigning type

```
>>> x = 1
>>> type(x)
<type 'int'>
>>> x = str(x)
>>> type(x)
<type 'str'>
>>>
```

# Language Basics

Variable

Math operator

Consider x = 5

| Operation | Example | Result |
|---|---|---|
| Addition | x = x + 5 | 10 |
| Subtraction | x = x - 5 | 0 |
| Multiplication | x = x * 5 | 25 |
| Division | x = x / 5 | 1 |
| Modulo | x = x % 2 | 1 |
| Exponent | x = x ** 2 | 25 |

# Table of Content

# Strings

## Strings Essentials

Strings are a collection of characters such as words and text…

“ABC”, “123” & “This is #1” are strings

```
>>> string1 = "hello"
>>> string2 = 'hello'
```

# Strings

## Style Format Strings

Text in a format string is simply printed

```
>>> print "I'd like %d %s" %(5, "cats")
```

# Strings

String Methods

```
>>> a = "string methods"

>>> dir(a)

['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__',
'__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'_formatter_field_name_split', '_formatter_parser', 'capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']

>>>
```

# Strings

## String Methods

Demo

Consider x = “pyGames”

| Operation | Example | Result |
|---|---|---|
| Upper case | `x.upper()` | PYGAMES |
| Lower Case | `x.lower()` | pygames |
| Title Case | `x.title()` | Pygames |
| Replace sub-string | `x.replace("G", "F")` | pyFames |
| Is sub-string in x | `"Games" in x` | True |
| Convert to List | `x.split()` | [‘pyGames’] |
| Length of String | `len(x)` | 7 |

# Strings

## Slicing Strings

Demo

| X = “ | I | | L | o | v | e | | K | M | A | “ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
| | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 | |

| Phép cắt chuỗi | Giá trị nhận được |
|---|---|
| X[0] | I |
| X[2] | L |
| X[0:3] hoặc X[:3] | I L |
| X[0:-1] hoặc X[:-1] | I Love KMA |
| X[0::2] hoặc X [::2] | Ilv M |
| X[::-1] | AMK evoL I |
| X[:6][::-1] | evoL I |
| X[5::-1] | evoL I |

# Strings

Encode/decode Strings

```
>>> x = "encode/decode demo"
>>> y = x.encode("base64")
>>> print y
ZW5jb2RlL2RlY29kZSBkZW1v

>>> y.decode("base64")
'encode/decode demo'
>>>
```

# Table of Content

# Functions

## Function Structure

```
1.  def function_name(argument1,argument2):
2.      'A Docstring for the help()' # Code Block 1
3.      # Code beneath it is executed when function is called
4.      # Arguments can be given default values by assigning them a value
5.          # Code block 2
6.          # Control statement of a loop, such as if/else or a for loop
7.      # The code can return one or more values
8.      return 'Return this string'
```

# Functions

## Exercises

1. pyGames question #02 - #11

# Table of Content

# if/else/elif

The “if” Statement

```
1.  if <logic expression>:
2.      # Code Block
```

# if/else/elif

## Logical Operators

| Toán tử | Ý nghĩa | Ví dụ |
|---|---|---|
| < | Nhỏ hơn | I < 100 |
| <= | Nhỏ hơn hoặc bằng với | I <= 100 |
| > | Lớn hơn | I > 100 |
| >= | Lớn hơn hoặc bằng với | I >= 100 |
| == | Bằng | I == 100 |
| != | Khác | I != 100 |
| not | Đúng khi biểu thức logic có kết quả sai | not I = 1 |
| and | AND logic | (I <= 9) and (X == True) |
| or | OR logic | (I > 3) or (F > 100.5) |

# if/else/elif

## Logic Truth Table

AND

False = 0

True = 1

| A | B | A AND B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# if/else/elif

Logic Truth Table

OR

False = 0

True = 1

| A | B | A OR B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# if/else/elif

The “if /else” & “elif” Statement

```
1.  if <logic expression>:
2.      # Code Block 1
3.      # Code Block 1
4.  else:
5.      # Code Block 2
6.      # Code Block 2
```

```
1.  elif <logic expression>:
2.      # Code Block
```

# Table of Content

# Lists

## Lists – So much more than Arrays

Lists are an indexed group of objects

Similar to arrays in other programming languages

```
>>> empty_list = []
>>> list_of_names = ['Alice', 'Bob', 'Eve']
```

Elements in the list are addressed based on their index

First ítem in the list have index 0 and go up from left to right

# Lists

## Lists – So much more than Arrays

```
>>>>>> new_list = [0] * 4
>>> len(new_list)
4
>>> new_list[3] = "Assignment to the last item."
>>> new_list
[0, 0, 0, 'Assignment to the last item.']
>>>
```

# Lists

## Lists – Methods

- `list[index] = value` : change an existing value
- `append(value)` : add an object to the end of the list
- `insert(position, value)` : insert the value at the given position
- `remove(value)` : remove the 1st matching item by its value
- `sort()` : sort the elements of the list
- `count(value)` : count occurrences of an item in the list
- `index(value)` : look up where a value is in the list
- `del list[index]` : delete an item by its index

# Lists

## Lists – Making a copy of list

```
>>> list_one = [ 1, 'one']

>>> wrong_copy_of_list_one = list_one

>>> copy_of_list_one = list(list_one)
```

# Lists

## Convert between Strings and Lists

- Convert Strings to Lists with `.split()`
- Convert Lists to String with `" ".join()`

# Lists

## Lists – `map()`

```
>>> def plus_1(a):
...     return a + 1
...
>>> map(plus_1, [0, 1, 100])
[1, 2, 101]
>>>
>>> def plus(x, y):
...     return x + y
...
>>> map(plus, [1, 2], [3, 4])
[4, 6]
>>>
```

# Table of Content

# For & While loop

## Loops

Loops are used to step through each element in Lists, Dictionaries and other iterable data structure

Examples:

```
for x in list:
for x in range(100):
for x in range(start, stop, step):
for index, value in enumerate(list):
while x:
```

# For & While loop

## Exercises

1. pyGames question #14 - #24

# Table of Content

# Dictionaries

Define

Loops are automatically indexed with Integer

With Dictionaries, user specify a “key” as the index

Dictionaries are very FAST to storing and retrieving data

```
>>> dict_one = {'first':'a', 'second':'b', 'third':'c'}
>>> dict_one['first']
'a'
>>>
```

# Dictionaries

## Copy of dictionary

```
>>> dict_one = {'first':'a', 'second':'b',
'third':'c'}
>>> copy_of_dict_one = dict(dict_one)
>>>
```

# Dictionaries

## Dictionaries – Methods

- `dict.items()` : returns a list of tuples with (key, value)
- `dict.key()` : returns a list of keys
- `dict.value()` : returns a list of values
- `"A" in dict`
- `"A" in dict.value()`

# Dictionaries

Loop through dictionary values

Loop using `.itervalues()` faster than `.value()`

```
>>> dict_one = {'a':'alpha', 'b':'beta'}
>>> for v in dict_one.itervalues():
...     print v
...
alpha
beta
>>>
```

# Dictionaries

Need both keys and values?

Loop using `.iteritems()` faster than `.items()`

```
>>> dict_one = {'a':'alpha', 'b':'beta'}
>>> for k, v in dict_one.iteritems():
...     print k, v
...
a alpha
b beta
>>>
```

# Dictionaries

## Exercises

1. pyGames question #25 - #28

# Table of Content

# Modules

## Install Modules

Using pip

- `pip list`
- `pip search <keyword>`
- `pip install <package>`
- `pip install --upgrade <package>`
- `pip uninstall <package>`

# Modules

## Using Modules

```
>>> import hashlib
>>> help(hashlib)

>>> dir(hashlib)
['__all__', '__builtins__', '__doc__', '__file__', '__get_builtin_constructor', '__name__', '__package__', '_ha
shlib', 'algorithms', 'algorithms_available', 'algorithms_guaranteed', 'md5', 'new', 'pbkdf2_hmac', 'sha1', 'sh
a224', 'sha256', 'sha384', 'sha512']
>>> print hashlib.md5.__doc__
Returns a md5 hash object; optionally initialized with a string
>>> dir(hashlib.md5('KMA'))
['__class__', '__delattr__', '__doc__', '__format__', '__getattribute__', '__hash__', '__init__', '__new__', '_
_reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'block_siz
e', 'copy', 'digest', 'digest_size', 'digestsize', 'hexdigest', 'name', 'update']
>>> type(hashlib.md5('KMA'))
<type '_hashlib.HASH'>
>>> type(hashlib.md5('KMA').digest)
<type 'builtin_function_or_method'>
>>> hashlib.md5('KMA').digest()
'\xb8P\xd6{Z\\\xd8A\x94\xd5\xfa\x8e\xdc\xc0\xabB'
>>> hashlib.md5('KMA').digest().encode('hex')
'b850d67b5a5cd84194d5fa8edcc0ab42'
>>> hashlib.md5('KMA').hexdigest()
'b850d67b5a5cd84194d5fa8edcc0ab42'
>>>
```

# Dictionaries

## Exercises

1. pyGames question #12 - #13