Python Setup

https://www.python.org/downloads

(install Python 3.5, not 2.7)

(check version with **`python --version`**)

github.com/samueltenka/train2/

(download example files for spam detection)

Introduction to Python

Michigan Hackers

Python Roadmap

2:40 Python: a Preview

2:55 Python v. C++
Python: a Textgame
A Quick Project

3:10 Python Containers,
Iterators, Functions

3:25 Spam Detection:
File IO and the
Math

3:40 Spam Detection:
Training and
Testing

3:55

What is Python?

- Interpreted
- Dynamically typed
- Versatile and Easy!
- Many common applications, including:
  - Text processing
  - Web development
  - Scientific programming
  - Machine Learning
  - Prototyping

Python: Two Ways to Code

Interact directly with Interpreter

- Best for small computations
- Does not require an entire executable file
- Fast and simple

Create .py file

- Best for larger scale projects
- Saves into .py file for later
- Can even be compiled

For now, let's use the standard IDE --- it's called IDLE

How Do I Write Python?

Python: First Program

```
print(“Hello World!”)
```

Python: a Preview

```
print(“Hello World!”)
```

Python's interpreted ---
try entering the above code
into IDLE

Python: a Preview

```
print(“Hello World!”)
```

Python's interpreted ---
try entering the above code
into IDLE

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", copyright", "credits" or "license" for more information.
>>>
```

Python: a Preview

```
print(“Hello World!”)
```

Python's interpreted ---
try entering the above code
into IDLE

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!")
```

Python: a Preview

```
print(“Hello World!”)
```

Python's interpreted ---
try entering the above code
into IDLE

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!")
Hello world!
>>>
```

Python: a Preview

```
4+5
```

We can also do arithmetic…

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> 4+5
```

Python: a Preview

```
4+5
```

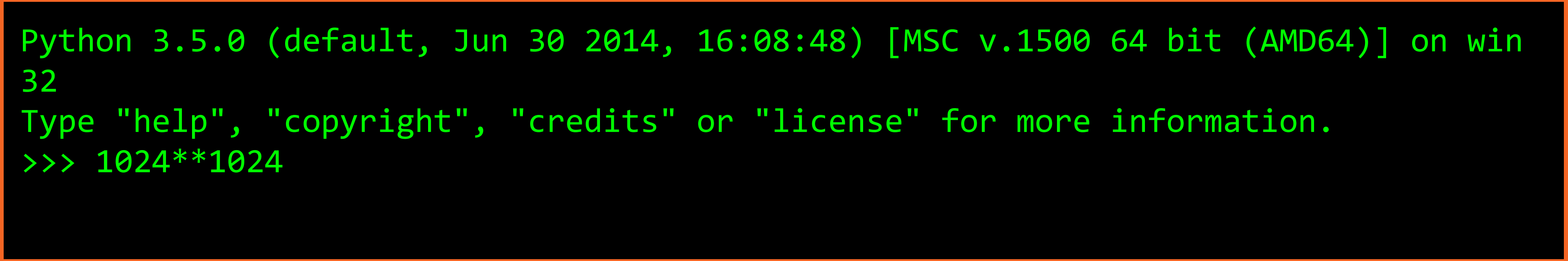
We can also do arithmetic...

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> 4+5
9
>>>
```

Python: a Preview

```
1024**1024
```

We can also do arithmetic that C++ can't do…

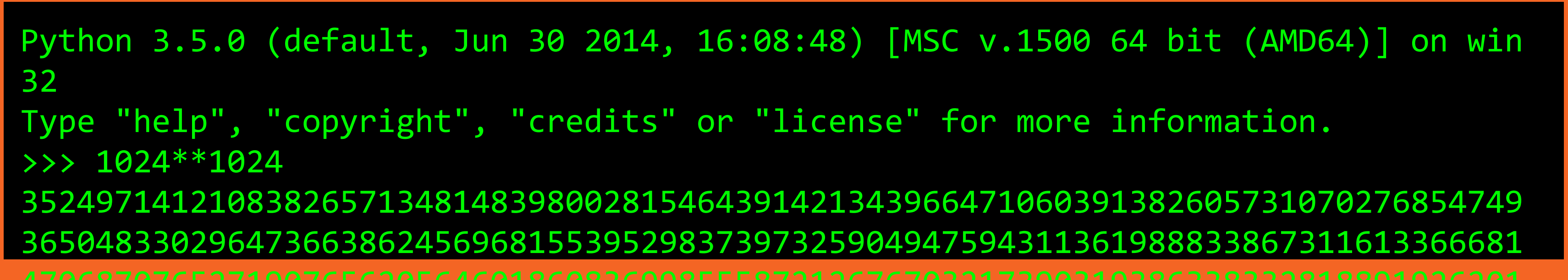


```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1024**1024
```

Python: a Preview

```
1024**1024
```

We can also do arithmetic that C++ can't do…



```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1024**1024
35249714121083826571348148398002815464391421343966471060391382605731070276854749
36504833029647366386245696815539529837397325904947594311361988833867311613366681
```

4706870765271907656205646018608369985558721267670321739031938633833281889192620 1
5842653180692314423926972687639995196119198034802329170347230576378241039458975 8
9345856311110781204353030326888187514464352913713571717556327753629326947950763 1
3436687469638004327689390246735321855830610856865924913760826763776003265851716 5
5733421064227734347575779978049902155982241243427508708431729345512957040670759 0
0020717046731355275335432173559875681076975779467857964124560483600729656168710 2
4866244650081059068183038134518514222987186837394598019859512993600379236190197 5
7683890508073335998909468700899941624772202006199255993140187235737970848858500 3
6669659306097304307741074074940180653658450770943205347006923544001698241315783 8
9153656916754682252425562742895026822086112236185768931940433324078692386463642 3
7802929158238455090401228426527712466745281698565933749758099159251020147976650 0
8774278345666191563143881075857435462890675510524340756781953453733639195713232 1
0113622615511765134329627207955793605376892875938357672870881305679305521293359 9
754278019219975348914740908681134673577843597833830910857175807228425031226776
9851973643594046830415066139436466661994548993636858018487767296858378032282161 1
3833854742443409221480450232563130417709625320794971672737737385983975520047739 9
7816512490691685793196090240739784153665765037875801240915720593951308532428243 9
2901089090690365154306903599631529865877499305168806703261450369876070529616967 8
1556418550966201822821857978020062536824015697620957222738065538832187097409859 5
0266919658902596119944875899737379297319172333554977239487887405085453278592247 5
8228364037939866231931740209314323814184370227604126822763829893548396254532412 8
980710826090513423467913095486757044735454976017469570785284527450279949438532
2948054451236883137876111968161671932763730814231510512052870468351518203832022 5
0786653139117317493642555621284434304945437214609406008640520972029509955435568 0 9

```
98519736435940468304150661394364666619945489936368580184877672968583780322821611
38338547424434092214804502325631304177096253207949716727377373859839755200477399
78165124906916857931960902407397841536657650378758012409157205939513085324282439
29010890906903651543069035996315298658774993051688067032614503698760705296169678
15564185509662018228218579780200625368240156976209572227380655388321870974098595
02669196589025961199448758997373792973191723335549772394878874050854532785922475
82283640379398662319317402093143238141843702276041268227638298935483962545324128
9807108260905134234679130954867570447354549760174695707852845274502799494385 32
29480544512368831378761119681616719327637308142315105120528704683515182038320225
07866531391173174936425562128443430494543721460940600864052097202950995543556809
48888157014704194108891565239711821728144232741409554280705943283816670482867719
72857703435525803544707834567774027206614143419982410109261930698311010857874866
84074385147285764533092916954840375108449472589372935545047377105998680105834202
19027353676279009748723681378389963973798981614548259709107328582027812829739376
428479733818386729806933990394293426130015951489680820516061022316242842367672
741265405434555310729662355960441332635214052961817117545065788425509933461872273
16979201855824371823913976733011681606825166392147065669814659617313748089491317
42364752993078326367714117001404210930251538132442219335072672096865184691303027
15696243977705370728658394976405515129181640254646245271913479717909921023357759
62779256460318241722748740845621134400433973951910654736207171042506860408965809
28700842593919173283844531470952205600874482302488523867074532907781264990865351
84468480701220803910828756453485450048639153887606361147666562023029481146835183
5374072060530215907909311281816131942219776L
```

```
98519736435940468304150661394364666619945489936368580184877672968583780322821611
38338547424434092214804502325631304177096253207949716727377373859839755200477399
78165124906916857931960902407397841536657650378758012409157205939513085324282439
29010890906903651543069035996315298658774993051688067032614503698760705296169678
15564185509662018228218579780200625368240156976209572227380655388321870974098595
02669196589025961199448758997373792973191723335549772394878874050854532785922475
82283640379398662319317402093143238141843702276041268227638298935483962545324128
980710826090513423467913095486757044735454976017469570785284527450279949438532
29480544512368831378761119681616719327637308142315105120528704683515182038320225
07866531391173174936425562128443430494543721460940600864052097202950995543556809
48888157014704194108891565239711821728144232741409554280705943283816670482867719
72857703435525803544707834567774027206614143419982410109261930698311010857874866
84074385147285764533092916954840375108449472589372935545047377105998680105834202
19027353676279009748723681378389963973798981614548259709107328582027812829739376
4284797338183867298069339903942934261300159514896808205160610223162428423676 72
74126540543455310729662355960441332635214052961817117545065788425509933461872273
16979201855824371823913976733011681606825166392147065669814659617313748089491317
42364752993078326367714117001404210930251538132442219335072672096865184691303027
15696243977705370728658394976405515129181640254646245271913479717909921023357759
62779256460318241722748740845621134400433973951910654736207171042506860408965809
28700842593919173283844531470952205600874482302488523867074532907781264990865351
84468480701220803910828756453485450048689153387606361147666562023029481146835183
53740720605302159079093112818161319422197 76L
```

'L' for '*long* integer'

Python: a Preview

```
pws2 = [2**i for i in range(10)]
```

We can make variables.

Any guesses what this does?

Python: a Preview

```
pws2 = [2**i for i in range(10)]
```

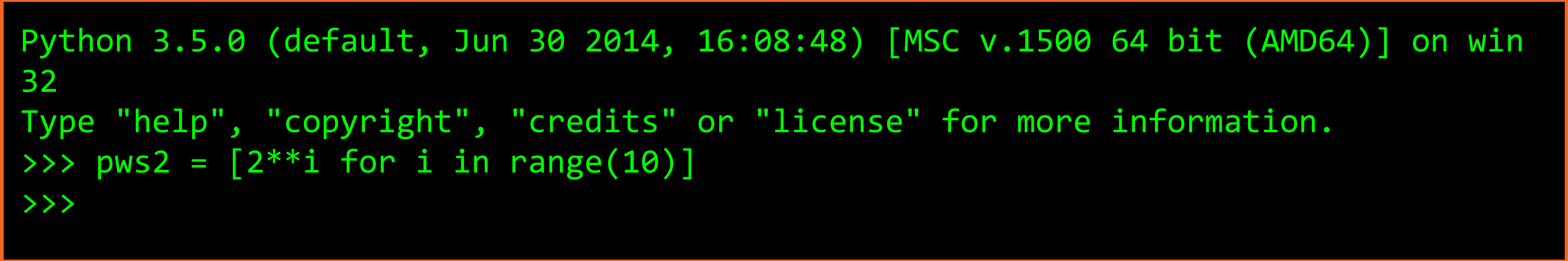
Here, we're finding
the first 10 powers of 2

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> pws2 = [2**i for i in range(10)]
```

Python: a Preview

```python
pws2 = [2**i for i in range(10)]
```

We can make variables.
Here, we're finding
the first 10 powers of 2



```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> pws2 = [2**i for i in range(10)]
>>>
```

Python: a Preview

```python
pws2 = [2**i for i in range(10)]
```

We can make variables.
Type the var. name to print:

```python
pws2
```

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> pws2 = [2**i for i in range(10)]
>>>
```

Python: a Preview

```
pws2 = [2**i for i in range(10)]
```

We can make variables.

Type the var. name to print:

```
pws2
```

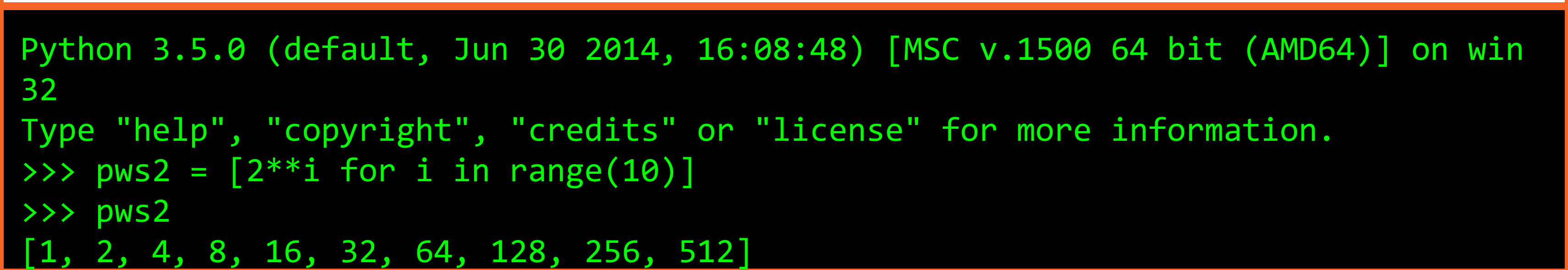
```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> pws2 = [2**i for i in range(10)]
>>> pws2
```

Python: a Preview

```
pws2 = [2**i for i in range(10)]
```

We can make variables.
Type the var. name to print:

```
pws2
```



```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> pws2 = [2**i for i in range(10)]
>>> pws2
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Python: a Preview

```
pws2 = [2**i for i in range(50)]
digs = [int(str(p)[0]) for p in pws2]
{d:digs.count(d) for d in range(10)}
```

Last example before we start learning this stuff systematically.
What do you think the 3 lines do? Try them!

Python: a Preview

```
pws2 = [2**i for i in range(50)]
digs = [int(str(p)[0]) for p in pws2]
{d:digs.count(d) for d in range(10)}
```

Line 1: finds the first 50 powers of 2
Line 2: finds those powers' first digits
Line 3: counts those digits' frequencies

Python: a Preview

```
pws2 = [2**i for i in range(50)]
digs = [int(str(p)[0]) for p in pws2]
{d:digs.count(d) for d in range(10)}
```

woahh, >30% of the powers of 2 start with the digit '1'!!!!

```
Python 3.5.0 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> pws2 = [2**i for i in range(50)]
>>> digs = [int(str(p)[0]) for p in pws2]
>>> {d:digs.count(d) for d in range(10)}
{0: 0, 1: 301, 2: 176, 3: 125, 4: 97, 5: 79, 6: 69, 7: 56, 8: 52, 9: 45}
>>>
```

C++ ⟷ Python

Python

```
print(“Hello World!”)
```

v. C++

```
#include<iostream>
int main(int argc,char** argv) {
    std::cout<<“Hello World!\n”;
}
```

Python

```
print(“Hello World!”)
```

v. C++

```
#include<iostream>
int main(int argc,char** argv) {
   std::cout<<“Hello World!\n”;
}
```

Python

```
print(“Hello World!”)
```

v. C++

```
#include<iostream>
int main(int argc,char** argv) {
   std::cout<<“Hello World!\n”;
}
```

Python

```
print(“Hello World!”)
```

v. C++

```
#include<iostream>
int main(int argc,char** argv) {
   std::cout<<“Hello World!\n”;
}
```

Python

Quickly read, write, and debug code
Programmer Friendly

v. C++

Quickly run code;
User Friendly

C++	⟷	Python	
`auto x = blah();`	⟷	`x = blah()`	Variable Initialization
`std::cout << “hey\n”;`	⟷	`print(“hey”)`	User Input/Output
`std::cin >> mystring;`	⟷	`mystring = input()`	
`if(b)`	⟷	`if b:`	Flow Control
`while(b)`	⟷	`while b:`	
`for(int i=0; i<5; ++i)`	⟷	`for i in range(5):`	
`for(auto x: xs)`	⟷	`for x in xs:`	

`FLOW CONTROL {`
`    // stuff`
`}`
	⟷	`FLOW CONTROL:`
		`    # stuff`	

Python: A Textgame

```
print(“Hi, who are you?”)
name = input()
if name == “Sam”:
   print(“You win!”)
while ‘y’ in input(“Crawl left?”):
   print(“You hurt your toe.”)
print(name, “falls into a hole.”)
```

Python: A Textgame

```
print(“Hi, who are you?”)
```

Python: A Textgame

```
print(“Hi, who are you?”)
name = input()
```

input() returns user input (everything you type until pressing ENTER).

Python: A Textgame

```
print(“Hi, who are you?”)
name = input()
```

no declaration needed; just initialization.

variables *are* typed --- in this case, “name” is a string.

but types are figured out at runtime.

Python: A Textgame

```
print("Hi, who are you?")
name = input()
if name == "Sam":
    print("You win!")
```

if condition:
  action

Python: A Textgame

```
print(“Hi, who are you?”)
name = input()
if name == “Sam”:
    print(“You win!”)
```

if condition:
  action

notice **colon** at end of line;
no braces, parentheses (or
semicolons).
Instead, code is organized by
indentation!

Python: A Textgame

```
print(“Hi, who are you?”)
name = input()
if name == “Sam”:
    print(“You win!”)
while ‘y’ in input(“Crawl left?”):
    print(“You hurt your toe.”)
```

while condition:
  action

Again, **colon** and **indentation**.

What’s the condition here?

Python: A Textgame

```
print(“Hi, who are you?”)
name = input()
if name == “Sam”:
    print(“You win!”)
while ‘y’ in input(“Crawl left?”):
    print(“You hurt your toe.”)
```

input(stuff) prints stuff,
then fetches user input.
We then check whether the
string ‘y’ was in that user input.
(‘y’ for ‘yes’, ‘yeah’, ‘yess!’, etc.)

Python: A Textgame

```
print(“Hi, who are you?”)
name = input()
if name == “Sam”:
   print(“You win!”)
while ‘y’ in input(“Crawl left?”):
   print(“You hurt your toe.”)
print(name, “falls into a hole.”)
```

Demo

Now It’s Your Turn!

Some Ideas (based on what you have learned thus far) :

Write a Chatbot:

```
q_responses = [‘yes’,‘probably?’,‘eww.’]
from random import choice
if ‘?’ in what_user_said:
      print(choice(q_responses))
```

Write a Find-the-Treasure Text Game:

```
x,y = 0,0
if input(‘where to?’) == ‘west’:
    x -= 1
if (x,y) == (4,2):
    # they won! print stuff and exit()
```

Container Types

Python Containers

```
mytuple = (1, 2, ‘3’)
mylist = [1, 2, ‘3’, ‘3’]
myset = {1, 2, ‘3’}
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
```

Python Containers

```
mytuple = (1, 2, ‘3’)
mylist = [1, 2, ‘3’, ‘3’]
myset = {1, 2, ‘3’}
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
```

Lists are ordered containers.
Random-access.

Python Containers

```
mytuple = (1, 2, ‘3’)
mylist = [1, 2, ‘3’, ‘3’]
myset = {1, 2, ‘3’}
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
```

Tuples are immutable, ordered containers. Random-access. Like a const list.

Python Containers

```
mytuple = (1, 2, ‘3’)
mylist = [1, 2, ‘3’, ‘3’]
myset = {1, 2, ‘3’}
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
```

Sets are unordered collections of unique objects. Hash-based.

Python Containers

```
mytuple = (1, 2, ‘3’)
mylist = [1, 2, ‘3’, ‘3’]
myset = {1, 2, ‘3’}
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
```

Dictionaries are unordered collections of key-value pairs. Hash-based. Like a set of x:y pairs, each x corresponding to one y.

Python Containers

```
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
mydict[‘one’]==1 #true statement
```

Dictionaries are unordered collections of key-value pairs. Hash-based. Like a set of x:y pairs, each x corresponding to one y.

Python Containers

```
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
mydict[‘one’]==1 #true statement
mydict[‘newkey’]==1 #runtime error
```

Dictionaries are unordered collections of key-value pairs. Hash-based. Like a set of x:y pairs, each x corresponding to one y.

Python Containers

```
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
mydict[‘one’]==1 #true statement
mydict[‘newkey’]==1 #runtime error
mydict[‘newkey’] = 0
```

Dictionaries are unordered collections of key-value pairs. Hash-based. Like a set of x:y pairs, each x corresponding to one y.

Python Containers

```
mydict = {‘one’:1, ‘two’:2, ‘three’:3}
mydict[‘one’]==1 #true statement
mydict[‘newkey’]==1 #runtime error
mydict[‘newkey’] = 0
mydict[‘newkey’]==1 #false statement
```

Dictionaries are unordered collections of key-value pairs. Hash-based. Like a set of x:y pairs, each x corresponding to one y.

C++	Python	
`const std::vector<void*> c = {&a, &b, &c};`	`x = tuple(a, b, c)`	Initialization
`std::string x;`	`x = str()`	
`std::vector<void*> x;`	`x = list()`	
`std::unordered_set<void*> x;`	`x = set()`	
`std::unordered_map<void*> x;`	`x = dict()`	
`*(x[key])`	`x[key]`	Indexing
`*(x[x.size()-index])`	`x[-index]`	
`//get all elements from i to j`	`x[i:j]`	
`std::find(x.begin(), x.end(), o)`	`x.find(o)`	Finding Value
`std::find(x.begin(), x.end(), o) != x.end()`	`o in x`	
`x.push_back(o);`	`x.append(o)`	Inserting Value
`x.insert(o);`	`x.add(o)`	
`x.insert(std::pair<kt,vt>(key, o));`	`x[key] = o`	

Python: Iteration

```
mylist = []
for i in range(10):
    mylist.append(i*i)
```

Python: Iteration

```
mylist = []
for i in range(10):
   mylist.append(i*i)
#the following are true:
mylist==[0,1,4,9,16,25,36,49,64,81]
mylist[5]==25
```

Python: Iteration

```
mylist = []
for i in range(10):
   mylist.append(i*i)
#the following are true:
mylist==[0,1,4,9,16,25,36,49,64,81]
mylist[5]==25
64 in mylist
65 not in mylist
```

Python: Iteration

```python
mylist = []
for i in range(10):
    mylist.append(i*i)
```

is the same as:

```python
mylist = [i*i for i in range(10)]
```

“List comprehensions”

Python: Iteration

```
mylist = []
for i in range(10):
    mylist.append(i*i)
```

is the same as:

```
mylist = [i*i for i in range(10)]
```

“List comprehensions”: Cleaner, more idiomatic.

Python: Iteration

```python
mydict = {}
for i in range(10):
    mydict[i] = i*i
```

is the same as:

```python
mydict = {i:i*i for i in range(10)}
```

“Dictionary comprehensions”

Python: Iteration

```
myset = set([])
for i in range(10):
    myset.add(i*i)
```

is the same as:

```
myset = set(i*i for i in range(10))
```

“Set comprehensions”

Python: Iteration

```
mytup = tuple([])
for i in range(10):
    mytup.append(i*i) #bad code! (why?)
```

is the same as:

```
mytup = tuple(i*i for i in range(10))
```

"Tuple comprehensions"

Questions?

Let's Create a More Complex Program: Spam Filter

SPAM

Dear Mr. samtenka01 Tenka:

I’m like a Nigerian Prince. I’ve run into some trouble with the Nigerian Princesses, and
as a result, I have the privilege of transferring $47,500,000 to your bank account. If you find this proposal acceptable, please send us your:

Bank Account Number
Facebook Username and Password
Class Schedule
A $50 processing fee.

Regards,
Prince Howgul Abul Arhu

VALID EMAIL

Dear Sam,

Attached is the paper we were discussing about Spam Detection --- let me know if you have any more questions.

Cheers,
Professor Lenhart

SPAM ...

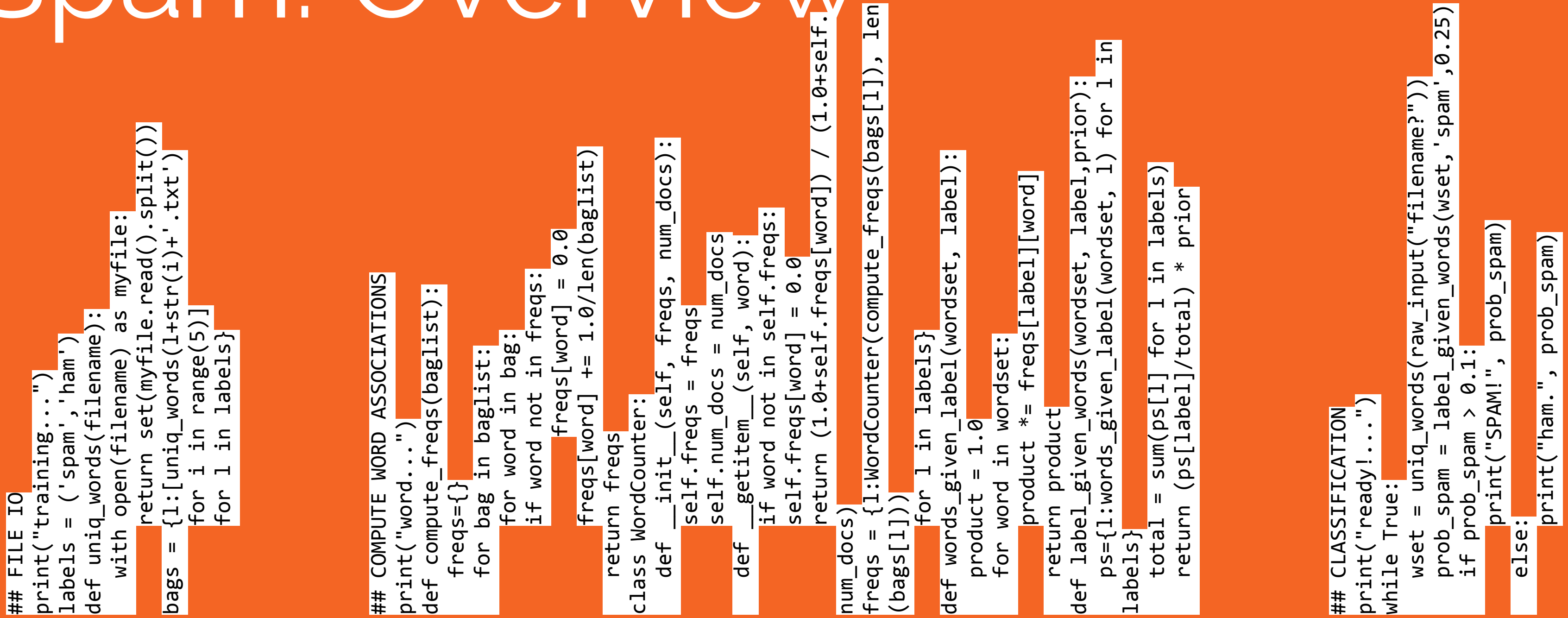
Dear Sam,

Attached is the paper we were discussing about Spam Detection --- let me know if you have any more questions.

Cheers,
Professor Lenhart

SPAM … OR HAM?

Spam: Overview



```
## FILE IO
print("training...")
labels = ('spam','ham')
def uniq_words(filename):
  with open(filename) as myfile:
    return set(myfile.read().split())
bags = {l:[uniq_words(l+str(i)+'.txt')
  for i in range(5)]
  for l in labels}
```

Read Training Samples

```
## COMPUTE WORD ASSOCIATIONS
print("word...")
def compute_freqs(baglist):
  freqs={}
  for bag in baglist:
    for word in bag:
      if word not in freqs:
        freqs[word] = 0.0
      freqs[word] += 1.0/len(baglist)
  return freqs
class WordCounter:
  def __init__(self, freqs, num_docs):
    self.freqs = freqs
    self.num_docs = num_docs
  def __getitem__(self, word):
    if word not in self.freqs:
      self.freqs[word] = 0.0
    return (1.0+self.freqs[word]) / (1.0+self.
num_docs)
freqs = {l:WordCounter(compute_freqs(bags[l]), len
(bags[l]))
  for l in labels}
def words_given_label(wordset, label):
  product = 1.0
  for word in wordset:
    product *= freqs[label][word]
  return product
def label_given_words(wordset, label,prior):
  ps={l:words_given_label(wordset, l) for l in
labels}
  total = sum(ps[l] for l in labels)
  return (ps[label]/total) * prior
```

Compute Word Associations

```
## CLASSIFICATION
print("ready!...")
while True:
  wset = uniq_words(raw_input("filename?"))
  prob_spam = label_given_words(wset,'spam',0.25)
  if prob_spam > 0.1:
    print("SPAM!", prob_spam)
  else:
    print("ham.", prob_spam)
```

Interactively Classify!

Spam Filter Part 1:

Reading a Text File (File IO)

Spam: Read Training Samples

```
with open('spample.txt') as myfile:
    text = myfile.read()
```

Spam: Read Training Samples

```python
with open('spample.txt') as myfile:
    text = myfile.read()
```

text == "Dear Mr. blah blah blah --- Dr. Mear"

Spam: Read Training Samples

```
with open('spample.txt') as myfile:
    text = myfile.read()
    words = text.split()
```

text == “Dear Mr. blah blah blah --- Dr. Mear”

words == [‘Dear’, ‘Mr.’, ‘blah’, ‘blah’, ‘blah’, ‘---’, ‘Dr.’, ‘Mear’]

Spam: Read Training Samples

```python
with open('spample.txt') as myfile:
    text = myfile.read()
    words = text.split()
    uniq_words = set(words)
```

text == "Dear Mr. blah blah blah --- Dr. Mear"

words == ['Dear', 'Mr.', 'blah', 'blah', 'blah', '---', 'Dr.', 'Mear']

uniq_words == {'Dear', 'Mr.', 'blah', '---', 'Dr.', 'Mear'}

Spam: Read Training Samples

```
with open('spample.txt') as myfile:
  uniq_words = set(myfile.read().split())
```

Spam: Read Training Samples

```
with open('spample0.txt') as myfile:
  uniq_words0=set(myfile.read().split())
with open('spample1.txt') as myfile:
  uniq_words1=set(myfile.read().split())
with open('spample2.txt') as myfile:
  uniq_words2=set(myfile.read().split())
...
```

Spam: Read Training Samples

```python
with open('spample0.txt') as myfile:
  uniq_words0=set(myfile.read().split())
with open('spample1.txt') as myfile:
  uniq_words1=set(myfile.read().split())
with open('spample2.txt') as myfile:
  uniq_words2=set(myfile.read().split())
...
```

what do you do to repeated code?

Spam: Read Training Samples

```python
def uniq_words(filename):
    with open(filename) as myfile:
        return set(myfile.read().split())
uw0 = uniq_words('spample0.txt')
uw1 = uniq_words('spample1.txt')
uw2 = uniq_words('spample2.txt')
...
```

Spam: Read Training Samples

```
def uniq_words(filename):
   with open(filename) as myfile:
      return set(myfile.read().split())
uw0 = uniq_words('spample0.txt')
uw1 = uniq_words('spample1.txt')
uw2 = uniq_words('spample2.txt')
...
```

what do you do to repeated code?

Spam: Read Training Samples

```python
def uniq_words(filename):
   with open(filename) as myfile:
      return set(myfile.read().split())
uws = []
for i in range(5):
   filename = 'spample'+str(i)+'.txt'
   uws.append(uniq_words(filename))
```

Spam: Read Training Samples

```python
def uniq_words(filename):
    with open(filename) as myfile:
        return set(myfile.read().split())
uws =[uniq_words('spample'+str(i)+'.txt')
        for i in range(5)]
```

Spam: Read Training Samples

```
def uniq_words(filename):
    with open(filename) as myfile:
        return set(myfile.read().split())
uws =[uniq_words('spample'+str(i)+'.txt')
        for i in range(5)]
```

```
uws == [{‘Dear’,‘Mr.’,...,‘Foreign’,‘Ambassador’,...‘Nigeria’,...},
        {‘Hello’,‘samtenka01’,...‘take’,...,‘short’,‘survey’,...},
        {‘Attention!’,...,‘buy’,‘our’,...}
        ...
       ]
```

Spam: Read Training Samples

```
def uniq_words(filename):
    with open(filename) as myfile:
        return set(myfile.read().split())
spambags=[uniq_words('spample'+str(i)+
            '.txt') for i in range(5)]
hambags=[uniq_words('hample'+str(i)+
           '.txt') for i in range(5)]
```

Spam: Read Training Samples

```
def uniq_words(filename):
    with open(filename) as myfile:
        return set(myfile.read().split())
spambags=[uniq_words('spample'+str(i)+
          '.txt') for i in range(5)]
hambags=[uniq_words('hample'+str(i)+
          '.txt') for i in range(5)]
```

what do you do to repeated code?

Spam: Read Training Samples

```python
def uws(filename):
   with open(filename) as myfile:
      return set(myfile.read().split())
bags ={label:[uws(label+str(i)+'.txt')
        for i in range(5)]
       for label in ('spample','hample')}
```

Spam: Read Training Samples

```
def uws(filename):
    with open(filename) as myfile:
        return set(myfile.read().split())
bags ={label:[uws(label+str(i)+'.txt')
          for i in range(5)]
         for label in ('spample','hample')}
```

```
bags == {‘spample’:[{‘Dear’,‘Mr.’,...,‘Foreign’,‘Ambassador’,...‘Nigeria’,...}
                    {...‘Piazza’...}, {...‘Perl’...}, ...],
         ‘hample’:[{‘Hey’,‘Sam’,...‘I’,‘liked’,‘your’,‘slides’,...,‘---’,‘Guido’}
                   {...}, {...}, ...]
        }
```

Spam Filter Part 2:

Training the Program

Spam: Word Associations

```
  freqs = {} #will contain number of word occurences:
  for wordset in list_of_wordsets:
     for word in wordset:
         if word not in freqs:
             freqs[word] = 0.0
         freqs[word] += 1.0
  for word in freqs: #normalize so everything adds to 1.0
    freq[word] /= len(list_of_wordsets)
```

Spam: Word Associations

```
def compute_freqs(list_of_wordsets):
  freqs = {}
  for wordset in list_of_wordsets:
     for word in wordset:
         if word not in freqs:
             freqs[word] = 0.0
         freqs[word] += 1.0
  for word in freqs:
    freq[word] /= len(list_of_wordsets)
  return freqs
```

```
freqs = {label:compute_freqs(bags[label])
         for label in ('spample','hample')}
```

Spam: Word Associations

```python
freqs = {label:compute_freqs(bags[label])
         for label in ('spample','hample')}
```

what do you do to repeated code?

Spam: Word Associations

```
labels = ('spample','hample')
freqs = {l:compute_freqs(bags[l])
         for l in labels}
```

Spam: Math

$$\text{Prob(spam|words)} = \frac{\text{Prob(words|spam)}}{\text{Prob(words|spam)} + \text{Prob(words|ham)}} \bullet \text{Prob(spam)}$$

$$\text{Prob(words|label)} = \text{Prob(word 1|label)} \bullet \text{Prob(word 2|label)} \bullet \ldots \bullet \text{Prob(word n|label)}$$

$$\text{Prob(word|label)} = \frac{\text{\#docs of type label containing word}}{\text{\#docs of type label, total}}$$

Spam: Math

$$\text{Prob(spam|words)} = \frac{\text{Prob(words|spam)}}{\text{Prob(words|spam)} + \text{Prob(words|ham)}} \bullet \text{Prob(words|spam)}$$

$$\text{Prob(words|label)} = \text{Prob(word 1|label)} \bullet \text{Prob(word 2|label)} \bullet \ldots \bullet \text{Prob(word n|label)}$$

$$\text{Prob(word|label)} = \frac{1 + \text{\#docs of type label containing word}}{1 + \text{\#docs of type label, total}}$$

"Laplace Smoothing": "expect the unexpected": handles words not present in our training examples.

Spam Filter Part 3: Determining if File is Spam

Spam: Classification

```
def words_given_label(wordset, label):
    product = 1.0
    for word in wordset:
        product *= freqs[label][word]
    return product
```

Spam: Classification

```
def words_given_label(wordset, label):
    product = 1.0
    for word in wordset:
        product *= freqs[label][word]
    return product
```

Problem: what if **`word`** is not a key of **`freqs[label]`**?

Spam: *Classification*

```
class WordCounter:
   def __init__(self, freqs, num_docs):
      self.freqs = freqs
      self.num_docs = num_docs
   def __getitem__(self, word): #<--comment
      if word not in self.freqs:
         self.freqs[word] = 0.0
      return (1.0+self.freqs[word]) / (1.0+self.num_docs)
```

Let's wrap the dictionary class in our very own "WordCounter" class. It'll automatically handle unknown keys and Laplace Smoothing

Spam: *Classification*

```
class WordCounter:
   def __init__(self, freqs, num_docs):
      self.freqs = freqs
      self.num_docs = num_docs
   def __getitem__(self, word): #<--comment
      if word not in self.freqs:
         self.freqs[word] = 0.0
      return (1.0+self.freqs[word]) / (1.0+self.num_docs)
```

Constructor

operator[] overload

First argument is always the class-instance the method belongs to. But when calling a method, specify other arguments only.

Spam: *Classification*

```
freqs = {l:compute_freqs(bags[l])
         for l in labels}
```

CHANGE

```
freqs = {l:WordCounter(compute_freqs(bags[l]),
                       len(bags[l]))
         for l in labels}
```

Spam: Classification

```
def label_given_words(wordset, label, prior):
  ps={label:words_given_label(wordset, l)
      for l in labels}
  total = sum(ps[l] for l in labels)
  return (ps[label]/total) * prior
```

Spam: Classification

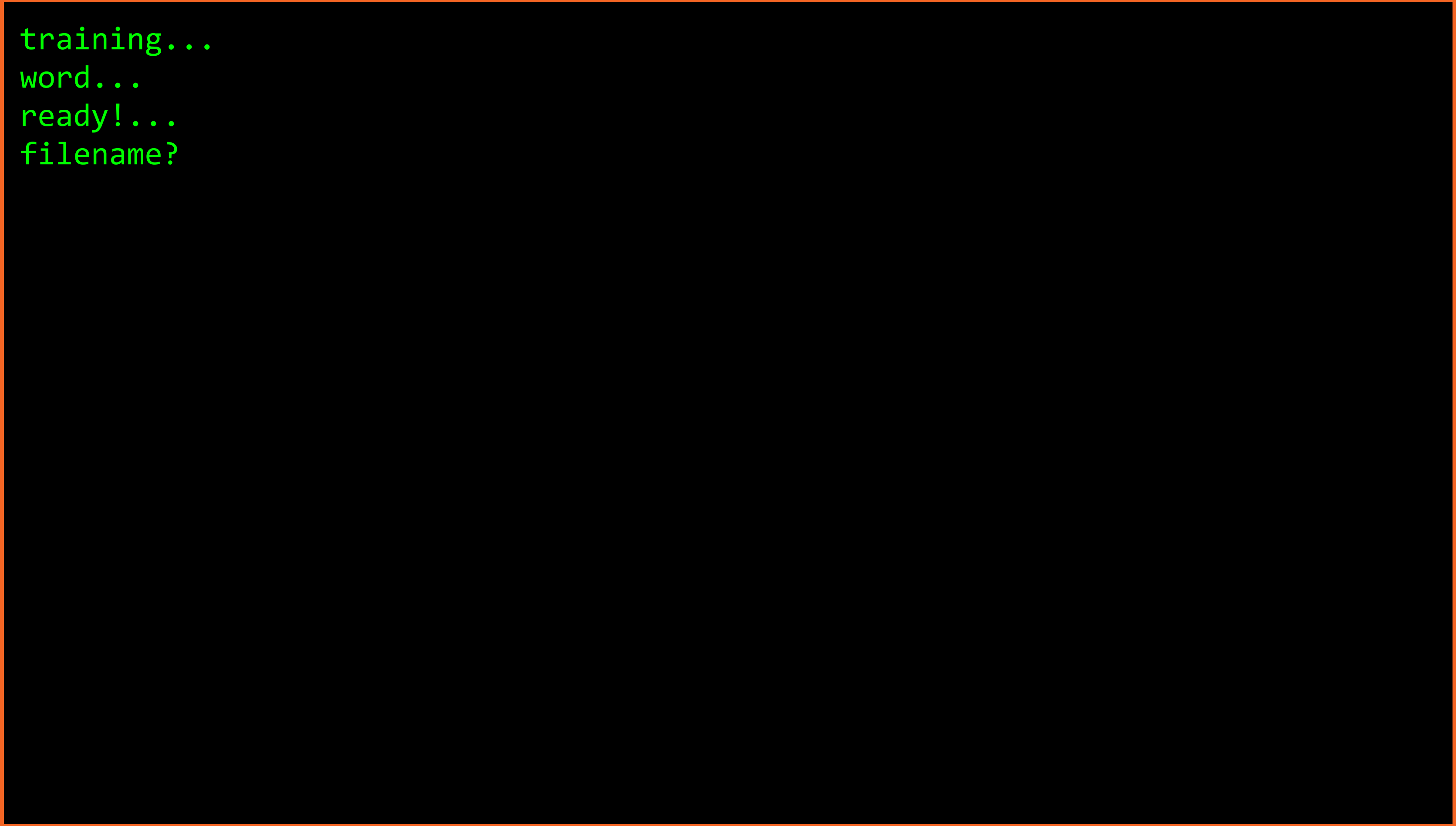
```
while True:
  wset = uniq_words(input("filename?"))
  prob_spam = label_given_words(wset,'spample',0.25)
  if prob_spam > 0.1:
      print("SPAM!", prob_spam)
  else:
      print("ham.", prob_spam)
```

Voila!
The Spam Filter is Done!

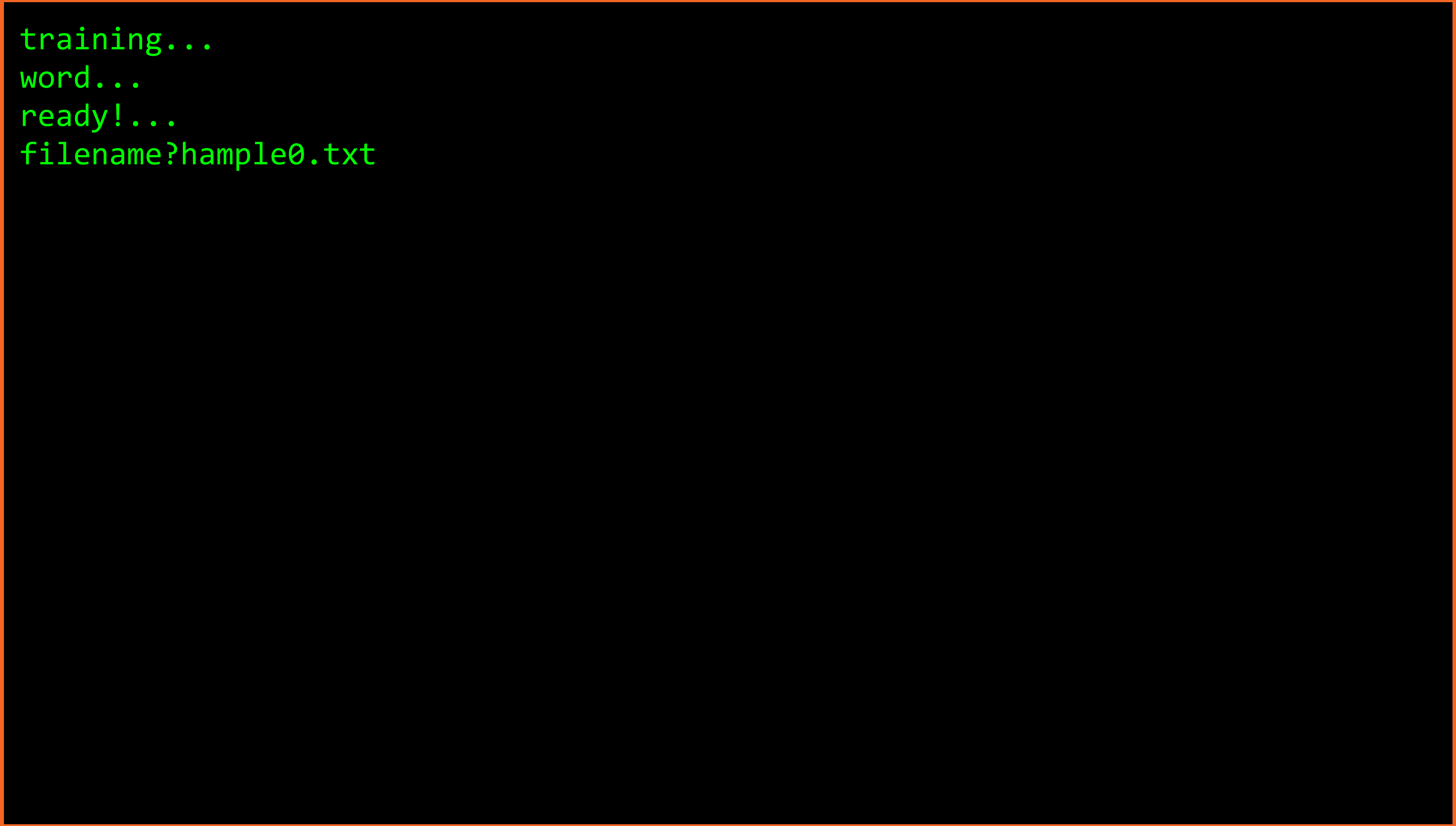
Voila!
The Spam Filter is Done!

Let's test…

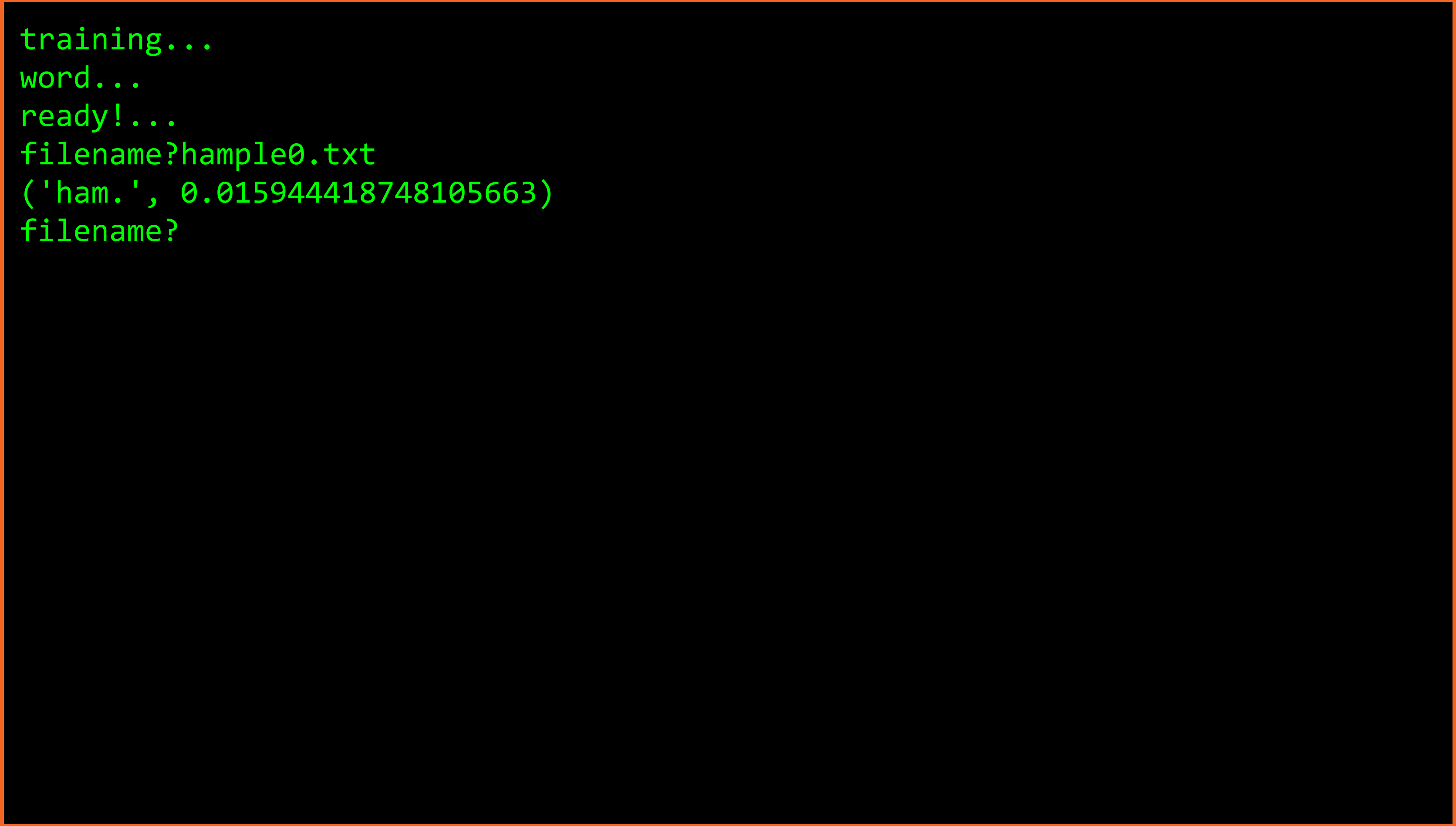
```
training...
```



```
training...
word...
ready!...
filename?
```



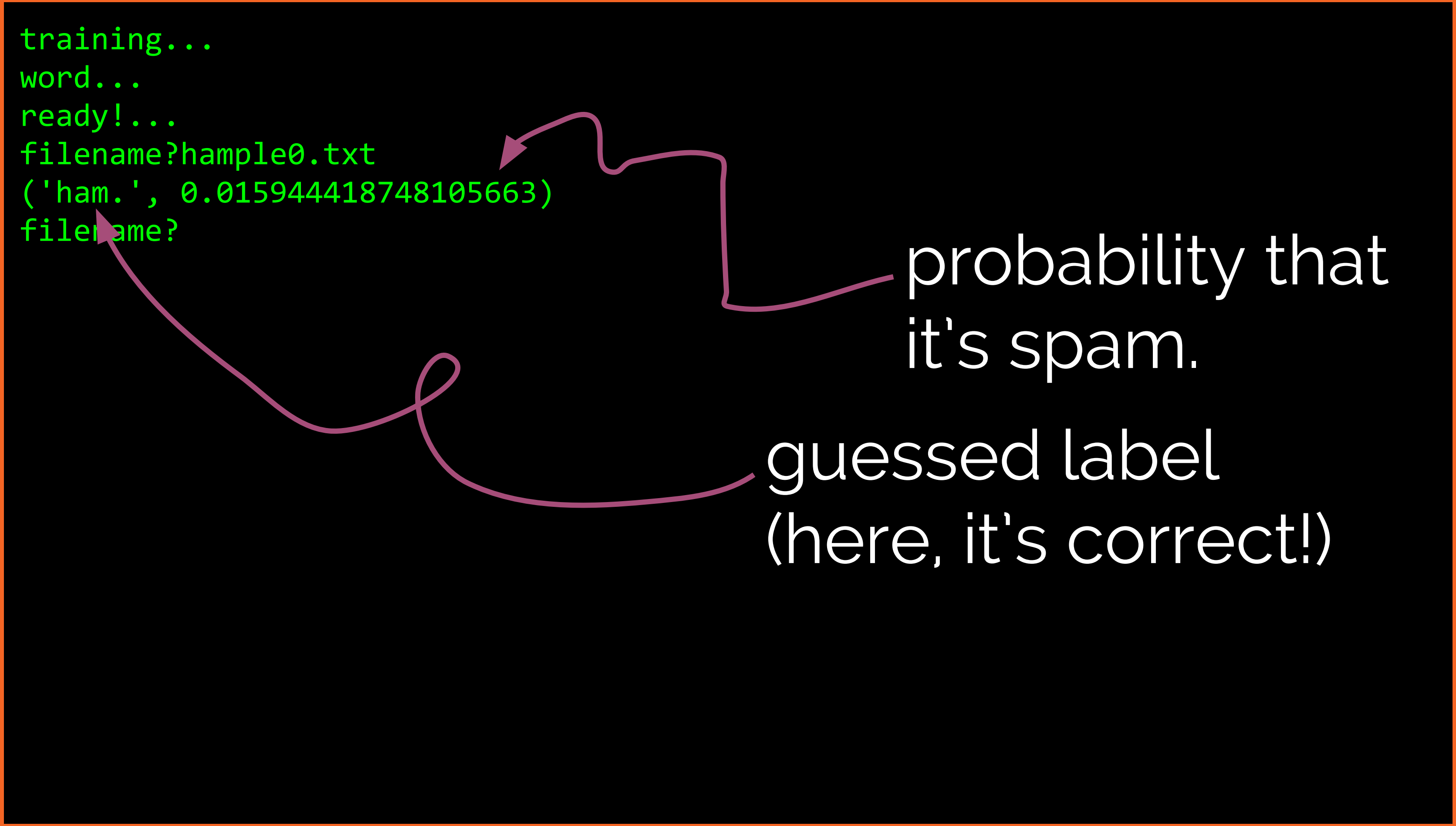
```
training...
word...
ready!...
filename?hample0.txt
```



```
training...
word...
ready!...
filename?hample0.txt
('ham.', 0.015944418748105663)
filename?
```

```
training...
word...
ready!...
filename?hample0.txt
('ham.', 0.015944418748105663)
filename?
```

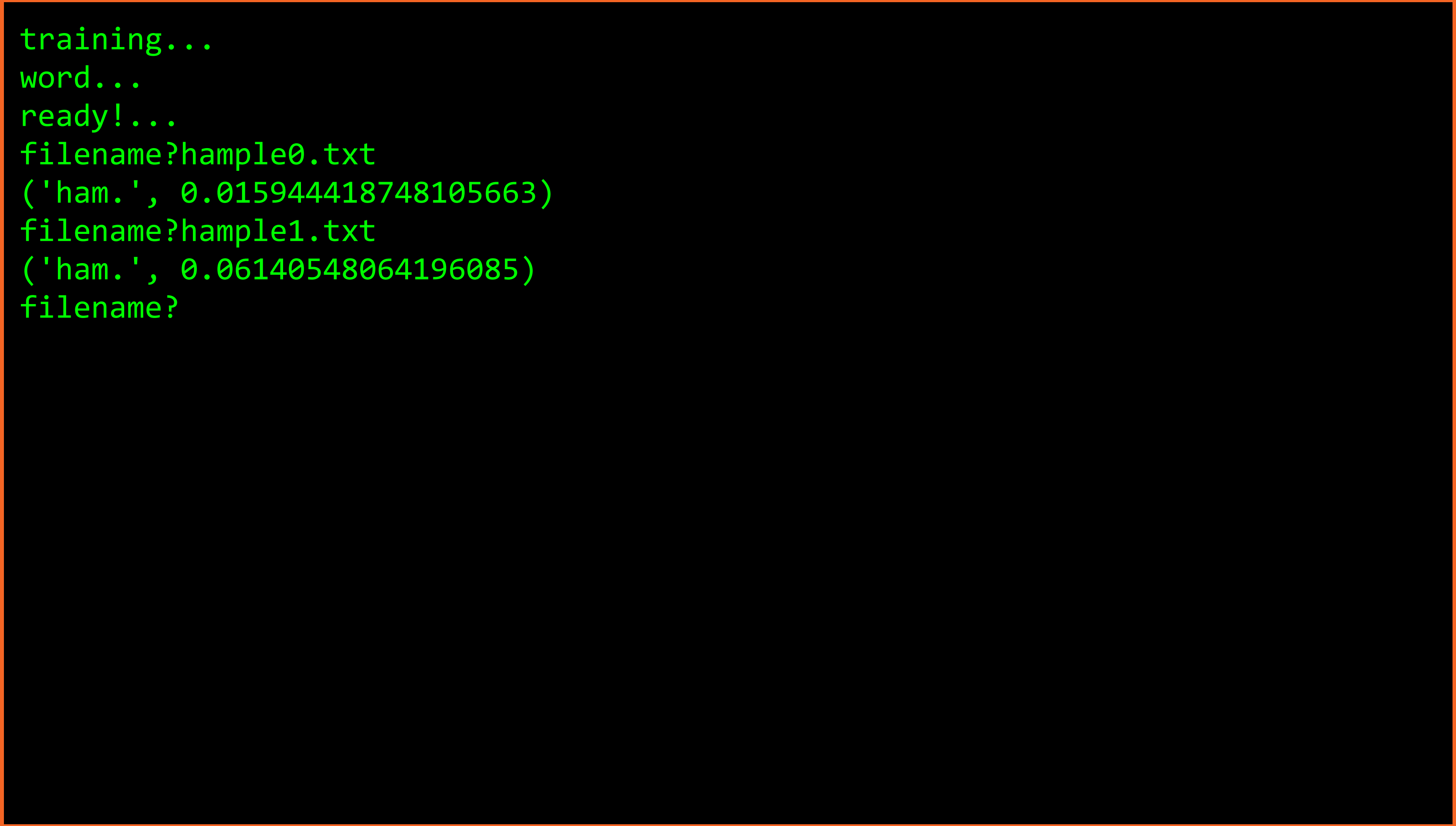
probability that it's spam.



```
training...
word...
ready!...
filename?hample0.txt
('ham.', 0.015944418748105663)
filename?
```

probability that it's spam.

guessed label (here, it's correct!)


```
training...
word...
ready!...
filename?hample0.txt
('ham.', 0.015944418748105663)
filename?hample1.txt
('ham.', 0.06140548064196085)
filename?
```

```
training...
word...
ready!...
filename?hample0.txt
('ham.', 0.015944418748105663)
filename?hample1.txt
('ham.', 0.06140548064196085)
filename?spample0.txt
('SPAM!', 0.2481797348775029)
filename?spample1.txt
('SPAM!', 0.2499866019220224)
filename?spample2.txt
('SPAM!', 0.2499866019220224)
filename?spample3.txt
('SPAM!', 0.24941626613843196)
filename?spample4.txt
('SPAM!', 0.24611646865081693)
filename?
```

all of these are correct!

```
training...
word...
ready!...
filename?hample0.txt
('ham.', 0.015944418748105663)
filename?hample1.txt
('ham.', 0.06140548064196085)
filename?spample0.txt
('SPAM!', 0.2481797348775029)
filename?spample1.txt
('SPAM!', 0.2499866019220224)
filename?spample2.txt
('SPAM!', 0.2499866019220224)
filename?spample3.txt
('SPAM!', 0.24941626613843196)
filename?spample4.txt
('SPAM!', 0.24611646865081693)
filename?
```

but we cheated: we're testing on the same examples as we trained on.

```
training...
word...
ready!...
filename?hample0.txt
('ham.', 0.015944418748105663)
filename?hample1.txt
('ham.', 0.06140548064196085)
filename?spample0.txt
('SPAM!', 0.2481797348775029)
filename?spample1.txt
('SPAM!', 0.2499866019220224)
filename?spample2.txt
('SPAM!', 0.2499866019220224)
filename?spample3.txt
('SPAM!', 0.24941626613843196)
filename?spample4.txt
('SPAM!', 0.24611646865081693)
filename?
```

but we cheated: we're testing on the same examples as we trained on. Can our program *generalize*?

spamtest.txt

Dear Mr. samtenka01 Tenka:

I’m like a Nigerian Prince. I’ve run into some trouble with the Nigerian Princesses, and
as a result, I have the privilege of transferring $47,500,000 to your bank account. If you find this proposal acceptable, please send us your:

Bank Account Number
Facebook Username and Password
Class Schedule
A $50 processing fee.

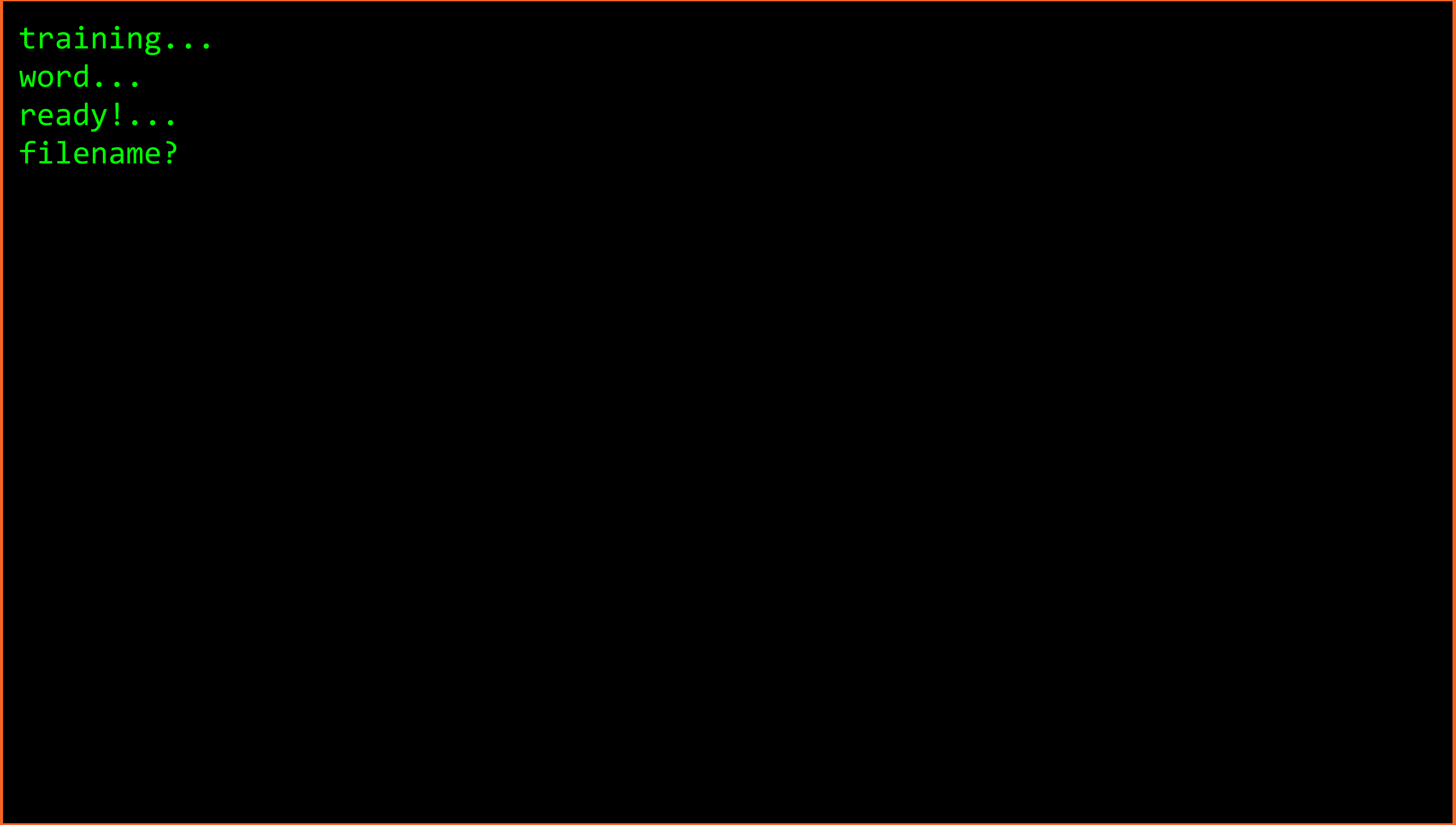
Regards,
Prince Howgul Abul Arhu

hamtest.txt

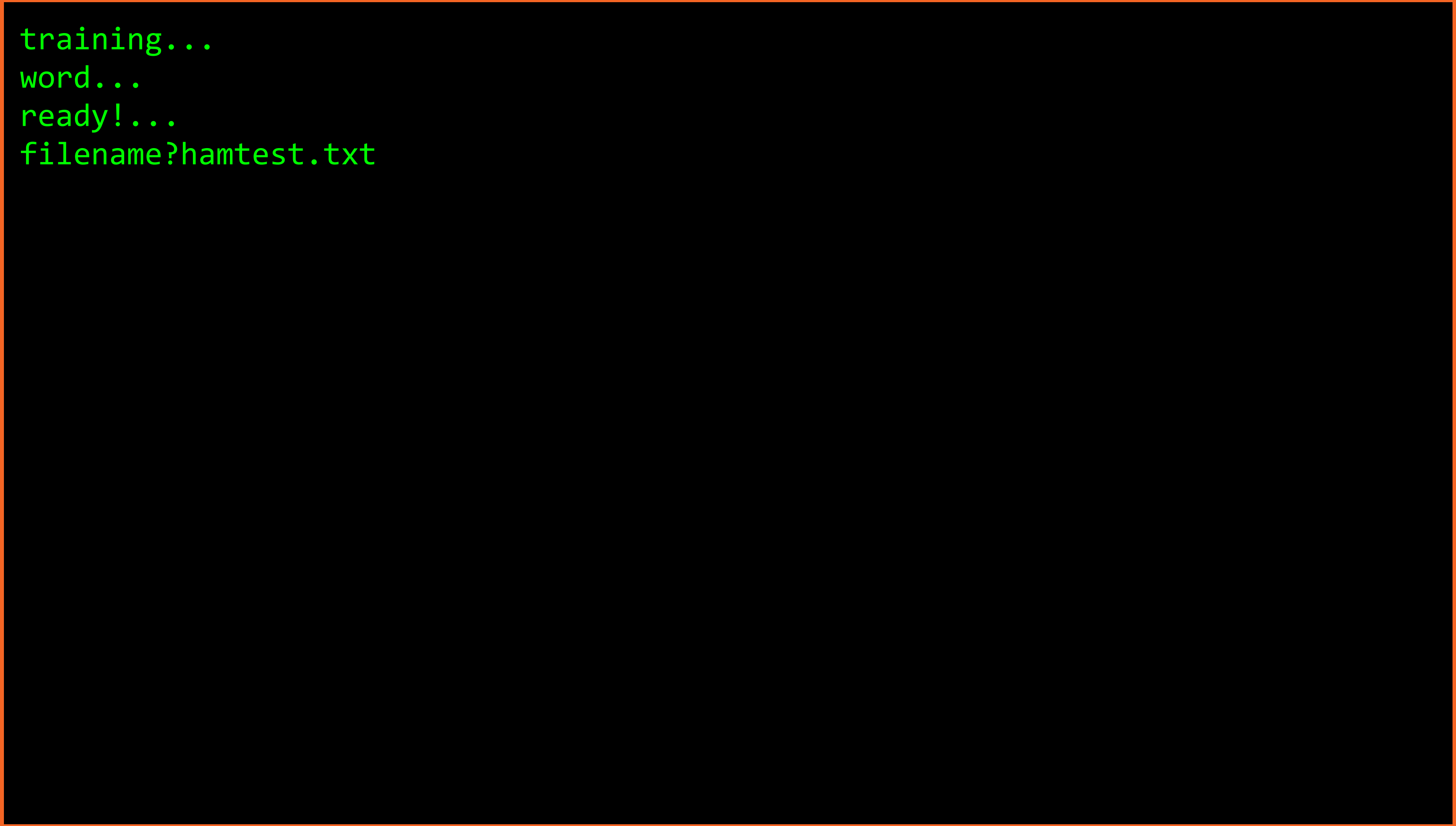
Dear Sam,

Attached is the paper we were discussing about Spam Detection --- let me know if you have any more questions.

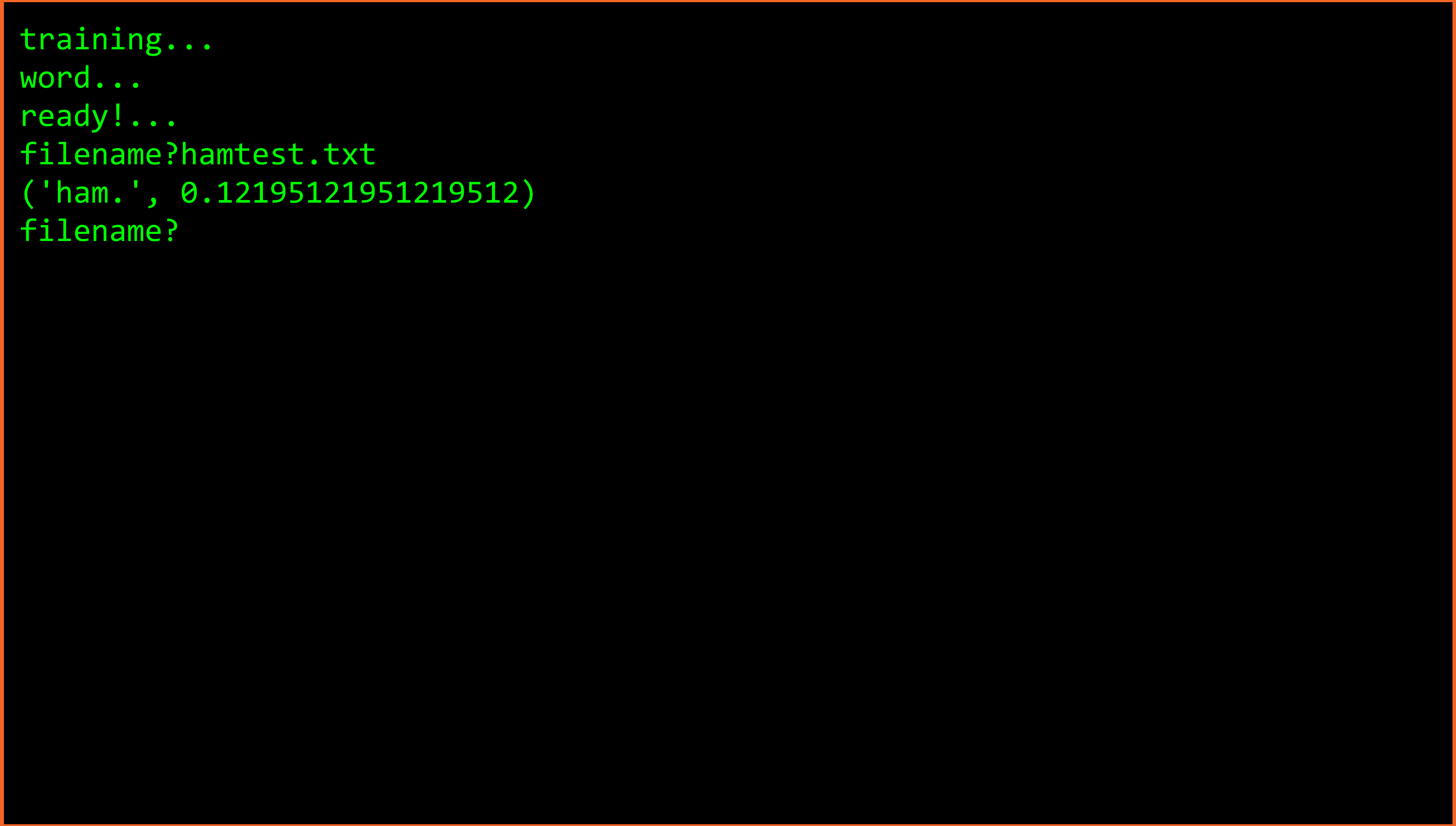
Cheers,
Professor Lenhart



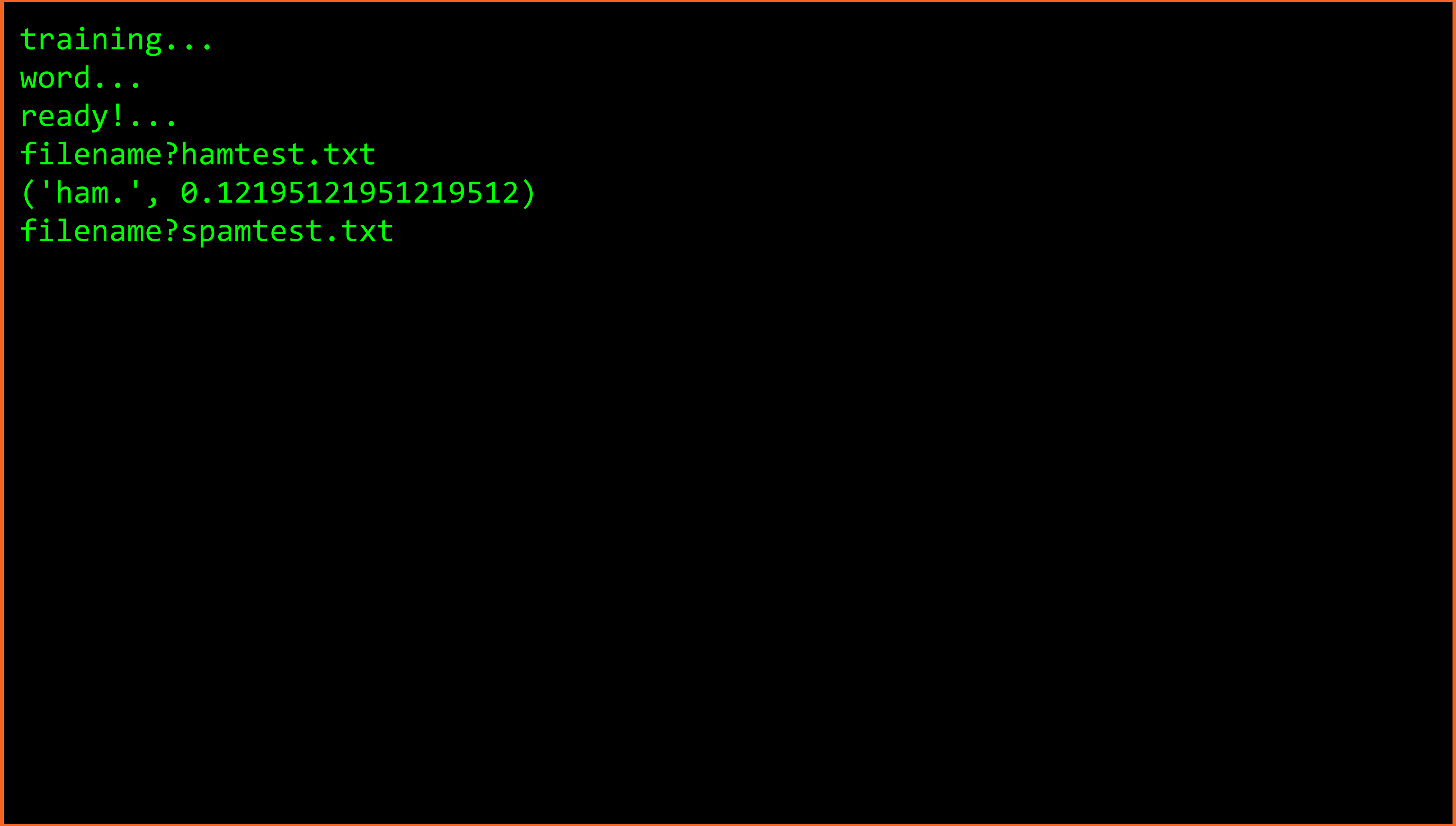
```
training...
word...
ready!...
filename?
```



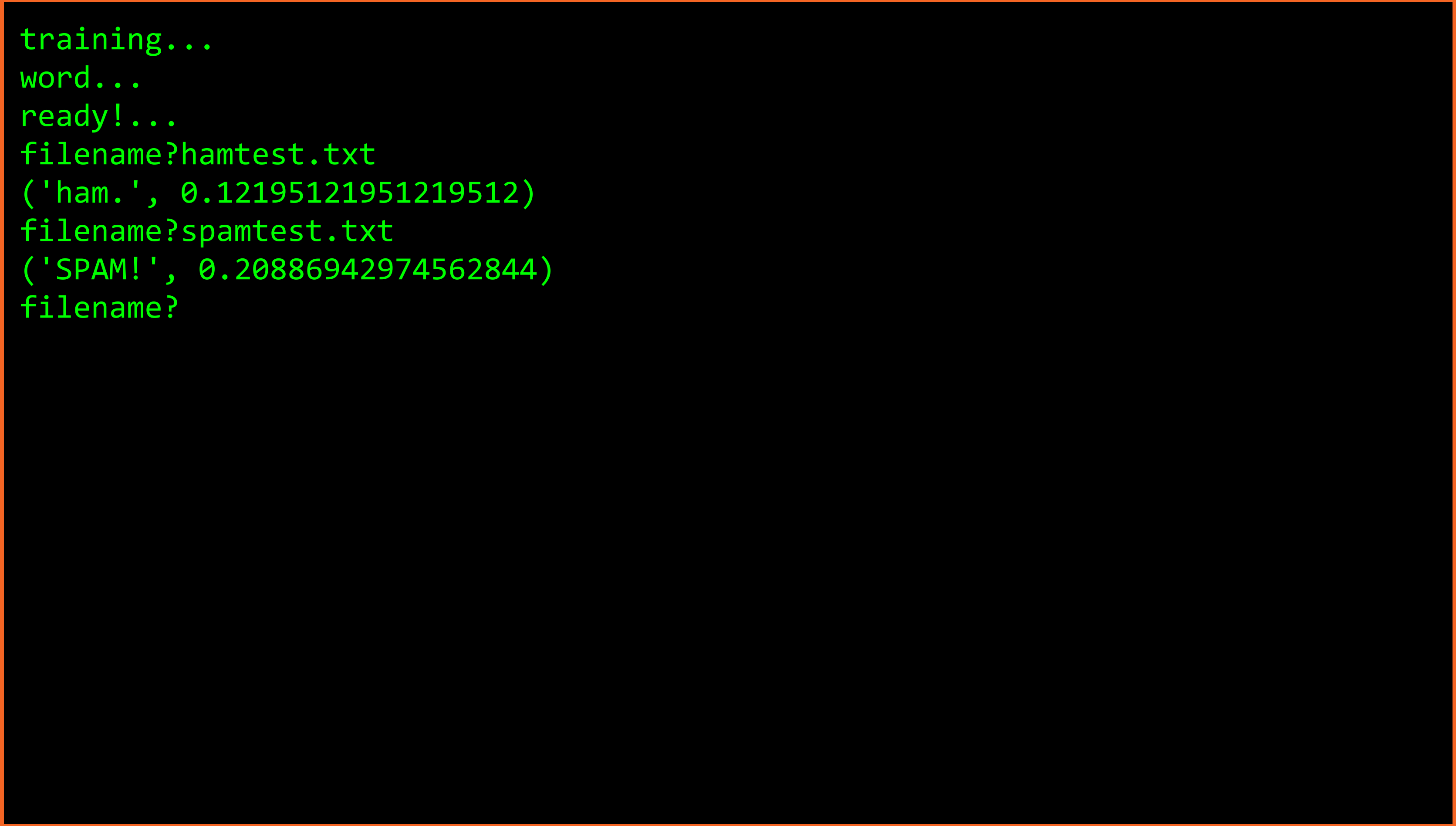
```
training...
word...
ready!...
filename?hamtest.txt
```



```
training...
word...
ready!...
filename?hamtest.txt
('ham.', 0.12195121951219512)
filename?
```



```
training...
word...
ready!...
filename?hamtest.txt
('ham.', 0.12195121951219512)
filename?spamtest.txt
```



```
training...
word...
ready!...
filename?hamtest.txt
('ham.', 0.12195121951219512)
filename?spamtest.txt
('SPAM!', 0.20886942974562844)
filename?
```

Voila!
The Spam Filter Works*!

* Of course, we still cheated: a real machine-learning application would train and test on 1000's of real-world examples. Our examples were hand-written to ensure success, not culled from my (surprisingly empty) spambox. The point was to demonstrate Python; but if you guys are interested in Machine Learning, I can highly recommend: Michigan's MSAIL Tutorials, and perhaps a future Machine Learning learn-to-hack!
(a learning experience both for programmer and program!)

Questions?

Python: More Resources

Official Documentation: https://docs.python.org/3/
Python Wiki, example code: https://wiki.python.org/moin/SimplePrograms

Python Packages:
--- Standard Library: https://docs.python.org/3/library/index.html
--- For Science! http://www.scipy.org/
--- For Games! http://www.pygame.org/download.shtml
--- For WebDev! https://www.djangoproject.com/ or
https://pypi.python.org/pypi/Flask
--- For Machine Learning! http://scikit-learn.org/stable/
--- More! http://goo.gl/OB0LM2

Wisdom from the BDFL: https://www.python.org/~guido/
Some cool tricks: http://goo.gl/pqT4pG

Python: More Stuff… ?

Classes

Python: More Stuff… ?

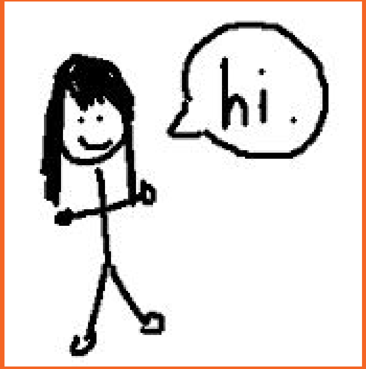
Classes

Functions

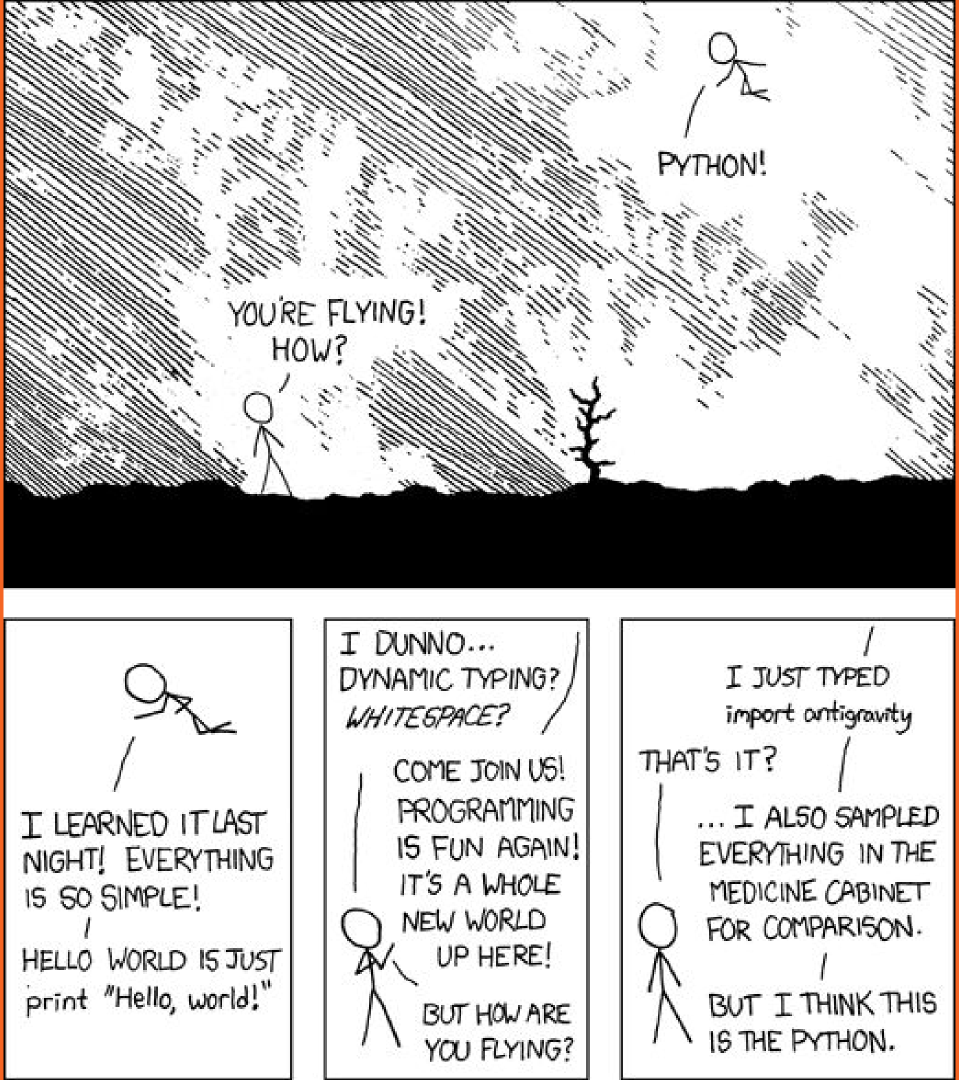
Python: More Stuff… ?

Classes

Functions



Regular Expressions


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.