HCT 2.0

كلّيات التقنية العليا
HIGHER COLLEGES OF TECHNOLOGY

CSF 2113
Programming for Information Security

Containers (CLO 1)

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

Abu Dhabi Men's College

Containers

- **Lists**
  A collection of values.

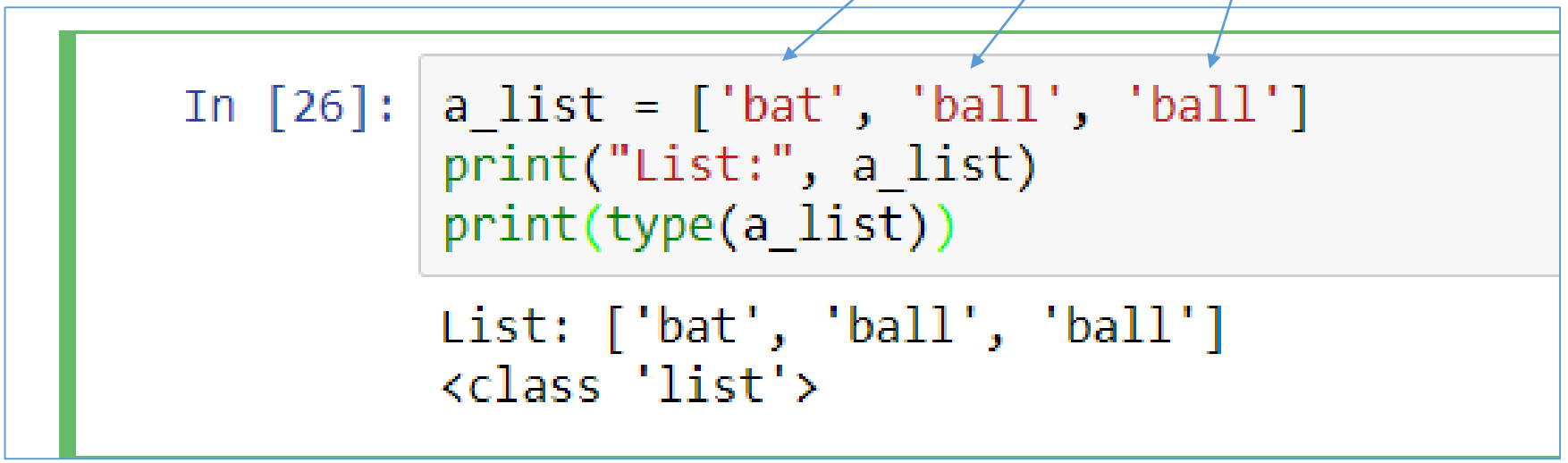
- **String data type**

- **Tuples**
  A collection of values as a record.

- **Dictionaries**
  A table of name, value pairs.

Lists

- A **list** is a sequence of values (objects).
- Individual values in a list are called **elements**.
- Lists have a **type** and are considered **values** themselves.
- To write a literal list (values)in your program, use the square brackets:



The **length** of a_list is 3 (the number of elements)

```
>>> print(len(a_list))    # gives 3 (an integer)
```

Working with lists

- Lists may contain elements of different types (though unusual)
- Elements are numbered starting from zero
  - As with strings, we call this the **position** or **index** of the element
- Accessing values in a list:
- To index from the last element u
- have to start by -1

Value	100	25	87	three	49	74	a	19
index	0	1	2	3	4	5	6	7
	-8	-7	-6	-5	-4	-3	-2	-1

```
numbers = [100, 25, 87, 'three', 49, 74, 'a', 19]
print("numbers[0]:",numbers[0])
print("numbers[1]:",numbers[1])
print("numbers[3]:",numbers[3])
print("numbers[6]:",numbers[6])
print("numbers[-1]:",numbers[-1]) # last index
```

```
numbers[0]: 100
numbers[1]: 25
numbers[3]: three
numbers[6]: a
numbers[-1]: 19
```

```
print("numbers[-2]:",numbers[-2])
print("numbers[-2]:",numbers[-3])
print("numbers[-2]:",numbers[-8])
```

```
numbers[-2]: a
numbers[-2]: 74
numbers[-2]: 100
```

List Slicing

Range: [start:stop:step] all optional

```
print("numbers", numbers)
print("numbers[0:4]:", numbers[0:4])  #numbers[start:stop]
print("numbers[:4]", numbers[:4])     #numbers[:stop]
print("numbers[4:]", numbers[4:])     #numbers[start:]
print("numbers[2:-2]", numbers[2:-2]) #numbers[start:stop]
```

```
numbers [100, 25, 87, 'three', 49, 74, 'a', 19]
numbers[0:4]: [100, 25, 87, 'three']
numbers[:4] [100, 25, 87, 'three']
numbers[4:] [49, 74, 'a', 19]
numbers[2:-2] [87, 'three', 49, 74]
```

Try numbers[0:100]

List element from index 0 to 100 should be shown.
Since the max index is 7, the whole will be shown without an error

```
print("numbers[0:9:2]", numbers[0:9:2]) #numbers[start:stop:step]
print("numbers[::2]", numbers[::2])     #numbers[::step]
                                        #reverse the list
print("numbers[::-1]", numbers[::-1])   #numbers[::step]
```

```
numbers[0:9:2] [100, 87, 49, 'a']
numbers[::2] [100, 87, 49, 'a']
numbers[::-1] [19, 'a', 74, 49, 'three', 87, 25, 100]
```

List mutability(editing of the list)

- Lists are mutable, they can be changed in several ways:
- Appending or inserting a new element
- Removing an element
- Sorting and reversing
- Changing the values of existing elements with assignment

```
numbers[0] = 300
print("numbers[0:]:", numbers[0:])
```

```
numbers[0:]: [300, 25, 87, 'three', 49, 74, 'a', 19]
```

Number[0}= 89
This will change the value of element in the index 0

Adding new things to a list

Add elements: The append method adds a new element to the end of a list

Append= to add one values one element

```
numbers = [100, 25, 87, 'three', 49, 74, 'a', 19]
# Add element at the end of list
numbers.append(300)
print(numbers)
```

The append method:

- Mutates the list
- Increases the length by one
- Does NOT return a value

gives [100, 25, 87, 'three', 49, 74, 'a', 19, 300]

Extending a list(for whole list)

- To add a whole list to the end, use the extend method
  - `scores = [80, 70, 60]`
  - `scores.extend([55, 88, 79])`
  - `# gives [80, 70, 60, 55, 88, 79 ]`
  - This increases the length by 3
  - Returns nothing!

- What if you used append instead of extend? (from previous slide)
  - `sc = [80, 70, 60]`
  - `sc.append ([55, 88, 79])`
  - Would give [80, 70, 60, [55, 88, 79]]
  - Adds the list as a single element
- The insert method adds a new element at a **given location**

  `sc.insert (2, "Rana")`
  - The new element will be at position 2 (that is, two elements to the left of it)
  - The other elements move to the right to make room, so no data is lost
  - [80, 70, "Rana", 60, [55, 88, 79]]

Remove elements from a list:

```
# remove a specific element, may raise exception
# if the element is not found
numbers.remove(300)
print("list after removing 300", numbers)
```

```
list after removing 300 [100, 25, 87, 'three', 49, 74, 'a', 19]
```

- This call will remove the first occurrence of 300 from the list numbers
- Exception/error if the element not found

Lets do it together...
Mathamatical operation = extends

```
numbers = [20, 15]
print(numbers)

numbers.extend([7,8])  # extends a list, in-place
print(numbers)

numbers = numbers + [9,10] # concatenate two lists
print(numbers)

numbers += [12,11] # another way to concatenate two lists
print(numbers)

#Append a list
numbers.append(40)
print(numbers)

numbers = numbers*2 # List elements twice
print(numbers)

numbers.sort() # Sorts the elements in a list
print(numbers)
```

Beware: list vs. element

```
[20, 15]
[20, 15, 7, 8]
[20, 15, 7, 8, 9, 10]
[20, 15, 7, 8, 9, 10, 12, 11]
[20, 15, 7, 8, 9, 10, 12, 11, 40]
[20, 15, 7, 8, 9, 10, 12, 11, 40, 20, 15, 7, 8, 9, 10, 12, 11, 40]
[7, 7, 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 15, 15, 20, 20, 40, 40]
```

Mutation versus making new objects

Notice that append, extend and insert return nothing!

- Most mutating functions in Python work this way
  - With a few exceptions later
- So do NOT do this
  - Numbers= numbers.append(255) = # ERROR – it will make numbers = None!
- Instead:
  - `numbers.append(255)  # GOOD – adds 255 at end`

Conversely, concatenating with + does NOT mutate the list

- Instead, it returns a new list
- So do NOT do this
  - `numbers + scores   # ERROR – does not store the new longer list`
- Instead:
  - `numbers = numbers + scores     #   or numbers += scores`

Searching in lists

There are two ways to search for an element in a list:

- You can use the `in` operator to check whether it's there:

```
>>>students = ["Tahir", "Ali", "Anas"]
>>>b = "Tahir" in students
>>>print(b)
```

- The `in` operator checks for that exact element, does not look "inside" elements

```
>>>"Hawk" in students          # False!
```

- To find an element's position in the list, use the index method

```
loc = students.index("Ali") # loc is 1
```

  - **An Exception/error is thrown if this element is not in the list**

Reversing a list

The `reverse` method reverses the order of a list

```
mylist = [“red”, “green”, “blue”]
mylist.reverse()
print(mylist)  #gives [“blue”, “green”, “red”]
```

- `reverse` **mutates the list!**
  - The original order is lost
- And `reverse` does not return a value
  - So do NOT do this:

```
mylist = mylist.reverse() # ERROR – mylist becomes None !
```

Reversing a list

- If you need a *new* reversed *copy* of the list, use

```
backwards = list(reversed(mylist))
```

  - For this one, mylist is unchanged, backwards is a new list

- Note the difference:
  - reverse is a method that mutates the list that is its argument
  - reversed is a function that returns a new sequence
    - It does not actually return a list, but you can convert it with **the list typecast**

Sorting a list

- You can also sort a list using the sort method
- Defaults to ascending order

```
scores = [ 75, 63, 92 ]
scores.sort()
print(scores)  # gives  [63, 75, 92]
```

- For strings, that means alphabetic (ASCII) order

```
physicists = [“Taher”, “Shahab”, “Husnain”]
physicists.sort()
print(physicists) # [“Husnain”, “Shahab”, “Taher”]
```

Sorting a list

- Can do descending order instead:

```
scores.sort(reverse = True)
print(scores) # gives [92, 75, 63]
```

- To make a new sorted list and keep the original list too

```
ordered_list = sorted(scores)
# does not mutate scores
```

  - `scores` is unchanged
  - `sorted` does make a new list
  - Similar to the difference between `reverse` and `reversed`

How to create a list

We've seen several different ways we can make a list:

- Hard code it: `lst = [ 1, 2, 3]`
- Start out with an empty list and append to it

```
lst = []
lst.append(1)
lst.append(2)
```

  - Useful as an accumulator

- Start out with an empty list and concatenate to it

```
lst  = []
lst += [1]
lst += [2]
```

How to create a list (cont'd)

- Split a string: `lst = "one two three".split()`
- Replication: `lst = [0] * 100`
  - Makes a list with 100 elements, each one zero
- Will have another when we get to external data files (readlines method)

Some more examples of lists

List	Description
[]	An empty list
[1,1,2,3,5,8]	A list of integers
[42, "What's the question?", 3.1415]	A list of mixed data types
["Stuttgart", "Freiburg", "München", "Nürnberg", "Würzburg", "Ulm","Friedrichshafen", Zürich", "Wien"]	A list of Strings
[["London","England", 7556900], ["Paris","France",2193031], ["Bern", "Switzerland", 123466]]	A nested list
["High up", ["further down", ["and down", ["deep down", "the answer", 42]]]]	A deeply nested list

2. String data type

There are different ways to define strings in Python:

```
>>> s = 'I am a string enclosed in single quotes.'
>>> s2 = "I am a string enclosed in double quotes."
```

Escaping with a backslash (\):

```
>>> s3 = 'It doesn\'t matter!’
 s3=“hii “how are u doing” “ check it in python
>>> s3 = "It doesn't matter!"    # no need of backslash

>>> txt = "He said: \"It doesn't matter\""
>>> print(txt)
```

He said: "It doesn't matter"

```
>>>
```

String data type

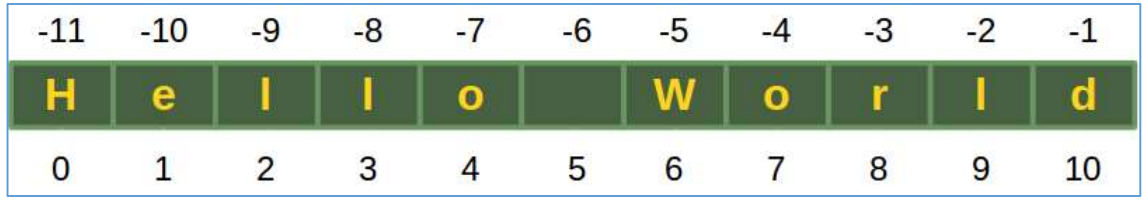
- Triple-quoted strings
  - unescaped newlines and quotes are allowed and retained

```
txt = '''A string in triple quotes can extend
over multiple lines like this one, and can contain
'single' and "double" quotes.'''
```

Extracting individual characters from a string

- String consists of a series or **sequence of characters** (letters, numbers, special characters, space, tab, etc)
- The characters in a string are numbered from 0 to length -1

```
>>> s = "Hello World"        # length = 11 (from 0 to 10)
```



- You can use square brackets to get the character at a given position
  - `first = s[0]                # this is “H”`
  - This is called subscripting or indexing
  - The position must be smaller than the length
    - `print(s[11])   # ERROR: string index out of range`

String length

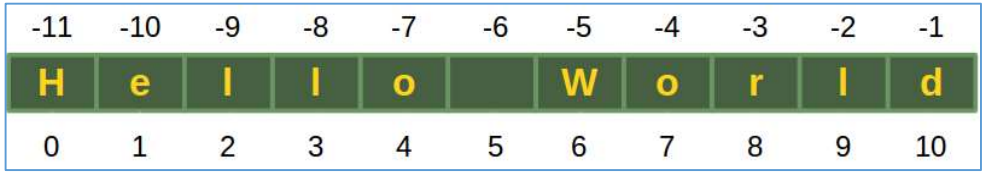
- The length of a string is the number of characters in it.
- Spaces count!
- To get the length of a string, use the `len` function:

```
s = "Hello World"
numchars = len(s)   # that's 11 characters
```

  - Argument type: string
  - Return type: integer
- What's len(“”)?
  - 0

Extracting characters with negative subscripts

- You can subscript with negative numbers, counting from the right end
  - `s[-1]` is the last, rightmost character
  - `s[-2]` is the next to last character
  - …
  - `s[-len(name)]` is the first, left most character



Strings are Immutable

- Python strings cannot be changed once created. That is strings are **immutable**

```
>>> s = "Some things are immutable!"
>>> s[1] = "."
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

Basic String Operations

Concatenation(only for String type)

"Hello" + "World" → "HelloWorld"

Repetition

String can be repeated or repeatedly concatenated with the asterisk operator "*":

"*-*" * 3 → "*-**-**-*"

Slicing

Substrings can be created with the slice or slicing notation:

"Python"[2:4] will result in "th"

Size

len("Python") will result in 6

Basic String Operations

- We can’t perform any other mathematical operations on strings, even if the strings look like numbers

- The following are **illegal**:
  - '2‘ - '1‘
  - 'eggs'/'easy'
  - 'third‘ * 'a charm‘

List and string

- A string is a sequence of characters and a list is **a sequence of values**, but a list of characters is not the same as a string. To convert from a string to a list of characters, you can use list:

```
>>> s = 'spam'
>>> t = list(s)
>>> t
['s', 'p', 'a', 'm']
```

```
s="spam"
print(s[0])
print(s[0:2])
print(s[-1])
print(s[::-1])
```

```
s
sp
m
maps
```

Strings versus lists

You've probably noticed a lot of similarities between lists and strings

- Sometimes the very same code works with both!
  - `len`, subscripts, slicing
  - Traversal
  - Concatenation
- But there are also many differences
  - Each element in a string is a character (a string of length 1)
  - But the elements of a list can be anything
    - Strings, numbers, bools, even other lists!
  - With strings, `in` searches for a substring
  - With lists, `in` searches for whole elements only
  - **Lists use `index` to locate an error (error if not found)**
  - Strings use `find` to locate a **substring (-1 if not found)**
- Another big difference: lists are **mutable**

```
b = 'Hell' in 'Hello World!'
print(b)
```

```
True
```

```
b = 2 in [1,2,3]
print(b)
b = 'Zayed' in ['Saeed','Zayed']
print(b)
```

```
True
True
```

```
word='Hello World!'
b = word.find('rld')
print("index:", b)
```

```
index: 8
```

```
x = [11,33,44,55]
b = x.index(55)
print("index:", b)
```

```
index: 3
```

3. Tuples

- Tuples are a collection of data items. They may be of different types. Tuples are *immutable* (like strings).
- Same as lists but Immutable

```
#Tuples
x = "bat", "ball", "ball"
print(x)
x = (1,)
print(x, type(x))
```

```
('bat', 'ball', 'ball')
(1,) <class 'tuple'>
```

- Python optionally uses brackets () to denote tuples
- We could have also used () for the above tuple
- If we have only one item, we need to use a comma to indicate it's a tuple: e.g. (“Bat”,)

Immutable Types

- Strings and Tuples are immutable, which means that once you create them, you can not change them.

- The assignment operator (=) can make a variable point to strings or tuples just like simple data types:

```
myString = “This is a test”

myTuple = (1,45.8, True, “String Cheese”)
          it contain numbers , bollean  and string)
```

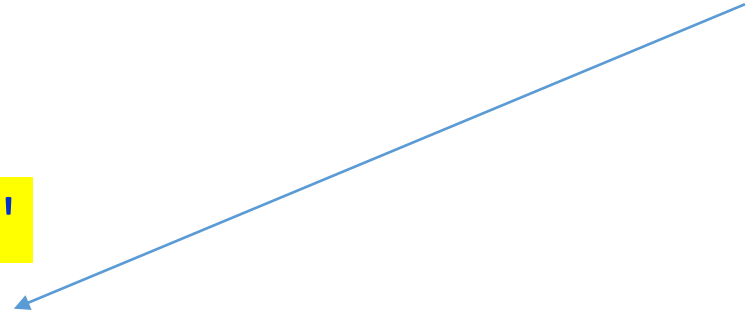
- Strings & Tuples are both sequences, which mean that they consist of an ordered set of elements, with each element identified by an index.

Working with Tuples

- Most list operators also work on tuples. The bracket operator indexes an element

```
>>> t = ("tuples", "are", "immutable")
>>> t[0]
'tuples'
```

Immutable Type



```
>>> t[0]="assignment not possible"
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Working with Tuples

- The slice operator selects a range of elements:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> t[1:3]
('b', 'c')
```

Try
- Range: [start:stop:step]
- + and *

```
rec1 = ("Name", "Address")
rec2 = ("Registration", "Phone")
rec = rec1 + rec2
print(rec)
new_rec = rec*2
print(new_rec)
```

```
('Name', 'Address', 'Registration', 'Phone')
('Name', 'Address', 'Registration', 'Phone', 'Name', 'Address', 'Registration', 'Phone')
```

Sorting tuples

```
>>> atuple = (33, 22, 11)
>>> atuple.sort()
Traceback (most recent call last):
…
AttributeError:
'tuple' object has no attribute 'sort'
>>> atuple = sorted(atuple)
>>> atuple
[11, 22, 33]
```

Tuples are immutable!

sorted( ) returns a list!

Most other things work!

```
>>> atuple = (11, 22, 33)
>>> len(atuple)
3
>>> 44 in atuple
False
```

Converting sequences into tuples

```
>>> alist = [11, 22, 33]
>>> atuple = tuple(alist)
>>> atuple
(11, 22, 33)
>>> newtuple = tuple('Hello World!')
>>> newtuple
('H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!')
```

List of Tuples

- Tuple is a record
- So a list of records could be constructed

```
rec1 = 'H001000', 'Saif Abdul Rehman', 'BSCS', 2.5, True
rec2 = 'H001001', 'Jane', 'BSCS', 2.0, False
rec3 = 'H001002', 'John', 'BSCS', 3.2, True

records = [rec1, rec2, rec3]
print(records)
```

```
[('H001000', 'Saif Abdul Rehman', 'BSCS', 2.5, True), ('H001001', 'Jane', 'BS
CS', 2.0, False), ('H001002', 'John', 'BSCS', 3.2, True)]
```

4. Dictionaries(Key: value)= pair

- A **dictionary** is like a list, but more general.
- In a list, the indices have to be integers; in a dictionary they can be (almost) any type.
- Keys must be ***unique*** within a dictionary: No ***duplicates***
- Simply put, a dictionary is a list of **key-value** **pairs.**

```
empty_dict = {}                                  # Empty dict Pythonic style
empty_dict2 = dict()                             # Empty dict less Pythonic
grades = { "Erik" : 80, "Tim" : 95 }             # dictionary literal
```

- Key and values are separated by a colon
- Paris of entries are separated by commas
- Dictionary is enclosed within curly braces

Keys: “Erik”, “Tim”
Values: 80, 95

Usage

- If we have

```
grades = { "Erik" : 80, "Tim" : 95 }
```

- Look up the value for a key using square brackets

```
grades["Erik"] is 80
grades[“Tim"] is 95
```

- **KeyError if you ask for a key that’s not in the dictionary**

```
kates_grade = grades["Kate"]
```

Check for the existence of a key using **in**:

```
erik_has_grade = "Erik" in grades      # True
kate_has_grade = "Kate" in grades      # False
```

Using get method to **return a value**:

```
eriks_grade = grades.get("Erik", 0)          # equals 80
kates_grade = grades.get("Kate", 0)          # equals 0
no_ones_grade = grades.get("No One")  # default is None
```

You assign key-value pairs using the same square brackets:

```
grades["Tim"] = 99           # replaces the old value
grades["Kate"] = 100         # adds a third entry
num_students = len(grades) # equals 3
```

Displaying contents

```
>>> age = {'Alice' : 25, 'Carol': 'twenty-two'}
>>> age.items()
dict_items([ ('Alice', 25), ('Carol', 'twenty-two')])

>>> age.keys()
dict_keys([ 'Alice', 'Carol'])

>>> age.values()
dict_values([25, 'twenty-two'])
```

Updating directories mutable

```
>>> age = {'Alice': 26 , 'Carol' : 22}

>>> age.update({'Bob' : 29})

>>> age
{'Bob': 29, 'Carol': 22, 'Alice': 26}

>>> age.update({'Carol' : 23})

>>> age
{'Bob': 29, 'Carol': 23, 'Alice': 26}
```

Dictionaries are mutable

```
>>> age = {'Alice' : 25, 'Bob' : 28}

>>> saved = age
>>> age['Bob'] = 29

>>> age
{'Bob': 29, 'Alice': 25}

>>> saved
{'Bob': 29, 'Alice': 25}
```

Dictionaries - a simple way to represent structured data:

```
tweet = {
"user" : “mhumayoun",
"text" : “Python is Awesome",
"retweet_count" : 100,
"hashtags" : ["#data", "#science", "#datascience"]
}
```

Besides looking for specific keys we can look at all of them:

```
tweet_keys      = tweet.keys()      # list of keys
tweet_values    = tweet.values()    # list of values
tweet_items     = tweet.items()     # list of (key, value) tuples
"user" in tweet_keys                # True, but uses a slow list in
"user" in tweet                     # more Pythonic, uses faster dict in
“mhumayoun" in tweet_values         # True
```

Summary of Containers or Compound Data Types

- A data type in which the values are made up of elements that are themselves values
- Strings – Enclosed in Quotes, holds characters, (immutable):
  “This is a String”
- Lists – Enclosed in square brackets, separated by commas, holds any data type (mutable):
  [4, True, “Test”, 34.8]
- Dictionaries – Enclosed in curly brackets. elements separated by commas. key : value pairs separated by colons. keys can be any immutable data type. values can be any data type (mutable):
  { 1 : “I”, 2 : ”II”, 3 : “III”, 4 : “IV”, 5 : “V” }
- Tuples – Values separated by commas, (usually enclosed by parenthesis) and can hold any data type (immutable):
  (4 , True, “Test”, 34.8)

كلّيات التقنية العليا
HIGHER COLLEGES OF TECHNOLOGY

CSF 2113
Programming for Information Security

CLO1: Iterations

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

Abu Dhabi Men's College

Loops / Iteration

- for loop
- range function
- Flow control within loops: break, continue, pass, and the “loop else”

While: Overview

We can use a while statement to repeatedly execute a series of tasks (i.e. a loop).

Syntax

initialization
while condition:
    statement(s)
    increment/decrement

Example

```
n = 0
while n < 10:
    print n
    n = n + 1
```

While: Control Flow

- Evaluate condition (which is a Boolean expression).
- If condition is **False**, then exit loop and continue program.
- If condition is **True**, then execute statements in body and then go to step 1.

```
n = 0
while n < 10:
        print n
        n = n + 1
```

While: Notes

- Each time through the body of a loop is called an iteration.
- The condition normally contains a variable that changes within the body of the loop so that we can eventually exit the loop.
- If a loop never exits, then this is called an infinite loop.

```
n = 0
while n < 10:
        print n
        n = n + 1
```

for loop: for ... in pattern

Used with collection data types which can be iterated through ("iterables"):

```
for i in [0,1,2] :
    print (i)
```

Output:
0
1
2

- Extract one **element** from the **list**.
- Execute the body of **for** loop with item bound to the **element**.
- Go to step 1.

```
for i in range(3):
    print(i)
```

```
0
1
2
```

The range function…

- range(n) returns a list of integers from 0 to n-1.
- range(0, 10, 2) returns a list 0, 2, 4, 6, 8

```
# Range function
x = range(0,4) # range(4)
print(x)

print(list(x))
print(tuple(x))
```

```
range(0, 4)
[0, 1, 2, 3]
(0, 1, 2, 3)
```

Similarly we have
str(), int(), float(), ….

Simple examples

- **Print each fruit in a fruit list:**

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

- **Looping Through a String**
  - Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

Nested loops

```
for x in range(0,3):
    for y in range(0,3):
        print(str(x) + ' * ' + str(y) + ' = ' + str(x*y))
```

```
0 * 0 = 0
0 * 1 = 0
0 * 2 = 0
1 * 0 = 0
1 * 1 = 1
1 * 2 = 2
2 * 0 = 0
2 * 1 = 2
2 * 2 = 4
```

Activity: Loop Exercise

- Find the smallest item in a list of numbers.

Flow control within loops

General structure of a loop:

```
while <statement> (or for <item> in <object>):
        <statements within loop>
        if <test1>: break       # exit loop now
        if <test2>: continue  # go to top of loop now
        if <test3>: pass        # does nothing!
else:
        <other statements> # if exited loop without
                                # hitting a break
```

Break

We can exit a loop immediately by using the **break** statement

```
import random

while True:
    r = random.randint(0, 10)
    if r == 0:
        break
    print("r inside loop with val",r)

print(r)
```

```
r inside loop with val 10
r inside loop with val 2
r inside loop with val 7
r inside loop with val 9
r inside loop with val 6
r inside loop with val 7
r inside loop with val 1
0
```

```
n=0
while n<10:
    if n==3:
        break
    print(n)
    n+=1
```

```
0
1
2
```

Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

Important to observe where do you break

Continue

We can skip the rest of the loop body and move on to the next iteration by using continue

```
n=0
while n<10:
    if n==3:
        n+=1
        continue
    print(n)
    n+=1
```

```
0
1
2
4
5
6
7
8
9
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

apple
cherry

Pass

- We do nothing
- Can act as a placeholder
  - A typical example: designing a new class with some methods that we don't want to implement yet

```
In [17]: n=0
         while n<10:
             if n==3:
                 pass
             print(n)
             n+=1
```

```
0
1
2
3
4
5
6
7
8
9
```

https://stackoverflow.com/questions/13886168/how-to-use-the-pass-statement-in-python

Using the “loop else”

An else statement after a loop is useful for taking care of a case where an item isn't found in a list:

In [40]:
```
for i in range(3):
    if i == 4:
        print ("I found 4!")
        break
    else:
        print ("Don't care about",i)
else:
    print ("I searched but never found 4!")
```

```
Don't care about 0
Don't care about 1
Don't care about 2
I searched but never found 4!
```

```
for i in range(5):
    if i==4:
        print("i found 4!")
        break

    else:
        print("dont care about
", i)
else:
    print("i searched but i
never found 4")
```

Activity: Guessing Game revisited

Write code that generates a random number between 0 and 100, and asks the user to guess what it is

Guess the number: 10
Guess a lower number!
Guess the number: 3
Guess a higher number!
Guess the number: 2
You got it!

- If the guess is too low, tell the user to guess 'Higher!'
- If the guess is too high, tell the user to guess 'Lower!'
- End the program when the user has found the number

```
import random
r = random.randint(0,10)
while True:
    Guess=int(input("Guess the number: "))
    if(Guess==r):
        print("You got it!")
        break
    elif(Guess<r):
        print("guess higher")

    else:
        print("Guess lower")
```

Output

```
Guess the number: 10
Guess lower
Guess the number: 2
guess higher
Guess the number: 6
Guess lower
Guess the number: 4
You got it!
```

كلّيات التقنية العليا
HIGHER COLLEGES OF TECHNOLOGY

HCT 2.0

CSF 2113
Programming for Information Security

CLO1: Functions and Modules

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

Abu Dhabi Men's College

The idea behind functions

- Sometimes we have to **repeat** the same combination of **control structures** and steps in several different places.

- **Functions** let you write the code once and use it at multiple places

- We've already seen several built-in and library functions
  - **print(…)**, **input**(…)

Calling a function

- And we know how to call them:
  - If it doesn't return a value:

```
func(arguments)
```

  - If it does return a value:

```
result = func(arguments)
```

Calling a function: syntax

- To call (“invoke”) a function, you write its name, followed by its **arguments** in parentheses ex. `func(arguments)`
- The name of the function is an identifier
  - **Same rules as for variable names**
  - **Letters, digits, underscore, cannot start with digit**
- The arguments are a list of expressions, separated by commas
  - Arguments are “inputs” that the calling statement sends to the function
  - Each function specifies how many arguments it takes
  - Some functions like random() take no arguments (**But they still need parentheses!)**

Calling a function: Semantics

- The interpreter **pauses** execution of the code that made the function call
  - But it remembers where it was
- The interpreter runs the code of the called function
  - It makes the arguments available to the function
- The function **finishes**/**returns**
  - The interpreter picks up where it left off
  - The value of the function call Is whatever value the function returned (or None if it does not explicitly return anything)

Structured programming guarantees

- When a subprogram (function) finishes, control returns to where the function was called from
- A function should have ONE return statement as the last line of the function definition (One entrance, one exit guarantee)
- or **NO return statement**, in which case all the statements in the definition are executed, then control returns to where it was called from

Defining a function

- To define a function, you need to specify **three** things:
  - The name of the function
  - What parameters the function takes
    - With a name for each parameter
  - What the function does when called
    - The code or body of the function
    - As usual, an indented block

Defining functions

```
def name (parameters):
    body
```

- Defines a function called “name”
- Goes at the **top level** of the file
  - Un-indented (at left margin)
  - Not inside `main` or other functions!
- Parameters are a comma-separated list of identifiers
  - **The function receives one argument for each parameter it has**

Defining functions

- When Python sees the definition, it does NOT run the body!
  - Instead it remembers it and remembers its name
  - The body runs when you call the function
  - If you don’t call a function, it will not run!
    - It does NOT run just because it is defined in the file
    - Something to check for when debugging

Functions

- Function definitions can contain any control structure you like.
- Ex: when square is called, the 3 is copied into the parameter `n`, **before** executing the body

What about with out returning satamtment

- Remember the Terminology:
  - `n` is the parameter
  - **3 is the argument (the value we give the parameter this time)**
- A function sends a value (or values) back to the caller by **returning** it (or them)

```
# A simple function

def square(n):
    """ returns the square of a number """
    return n*n

print ("The square of 3 is", square(3))
```

```
The square of 3 is 9
```

Now when you call `square(val)`, it prints the `square` of that `val`

Body of the function should be indented consistently (4 spaces is typical in Python;
Beware of tabs -- AVOID THEM)

Docstring: Shift + Tab(information about the function)

Place your cursor inside function name and press "Shift+Tab"

```
square(2)
```

```
Signature: square(n)
Docstring: returns the square of a number
File:      c:\users\mhumayoun\pycode\<ipython-input-1-4ddd1422cfe7>
Type:      function
```

Alternatively:

```
In [5]: ?square()
```

```
Signature: square(n)
Docstring: returns the square of a number
File:      c:\users\mhumayoun\pycode\<ipython-input-1-4ddd1422cfe7>
Type:      function
```

Function variables are local

- Variables declared in a function do not exist outside that function

```
#Function variables are local
def square(n):
    m = n*n
    return m

print ("The square of 3 is", square(3))
print (m)
```

```
The square of 3 is 9

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-eb29a3b46de1> in <module>()
      5
      6 print ("The square of 3 is", square(3))
----> 7 print (m)

NameError: name 'm' is not defined
```

- They are **local**
- For example, `m` and `n` are local variables in `square` function
- The locals only exist while the function is running
  - **Lifetime** or **extent:** the time during which a variable takes up memory
  - Variable is “born” when the function is initialized…
  - … and “dies” when the function it is in returns

Local variables

- Other functions cannot see local variables at all!
  - The scope of the variable is which part of the code can see it
  - The scope of a local variable is the body of the function it is in, starting from its initialization
- This means your function cannot refer directly to variables in other functions.
- If you need access to information from another function
  - Use a parameter and argument to send information into a function
  - Use the return value to get information back out
- A global variable is defined outside any function
  - **Avoid these!** Very hard to reason about, they can be changed by ANY function at ANY time!

Scope example
a= global
c= global try with out return statement in the funcation

```
#Scope example
a = 5                  # global
def func(b):
    global c
    c = a + b
    return c

print (func(4))
print (c)
```

```
9
9
```

Functions with multiple parameters

- Functions can have more than one parameter
- When calling the function, you must supply the same number of arguments as parameters

```
power(5)  #ERROR
power(5,3,4) #ERROR
```

```
def power(val, p):
    """ returns the pth power of a number val"""
    return val**p

x,p = 9, 2
print(x, "raise to power", p, "is", power(x,p))
```

```
9 raise to power 2 is 81
```

- If arguments are in wrong **order**, or if arguments are the wrong **type**, the function will eventually give an error

```
power("Hello", "this") #ERROR
```

Functions without return values

- All functions in Python return something.
- If a return statement is not given, Python returns **None**
- For example, the list append function changes the list but does not return a value: funcation, sort , reverse,doesnot have return
-

```
a = [0, 1, 2]
b = a.append(3)
print (b)
```

Output

None

Parameters/return values vs. input/output

- parameters take “input” from the statement that calls the function
- return value send “output” to the statement that called the function
- input() takes input from the keyboard (the user)
- print() sends output to the screen (the user)
- Good functions usually use parameters and return a value,
  - Good functions do not usually use input and print
  - Then you can use them not only with user input…
  - … but also with graphical programs, files, computed data…
- ***When should a function use input or print?***
  - When the function’s sole purpose is to interact with the user

كلّيات التقنية العليا
HIGHER COLLEGES OF TECHNOLOGY

CSF 2113
Programming for Information Security

Fundamentals of File Handling

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

Abu Dhabi Men's College

I/O in Python: Reading a file

- Python provides basic functions and methods necessary to manipulate files by default using a **file** object.
- The *open* Function: Before you can read or write a file, you have to open it using Python's built-in *open()* function.
  - This function creates a **file** object, which would be utilized to call other support methods associated with it.

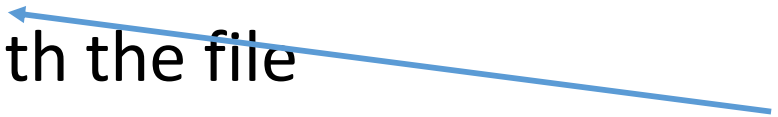
```
>>>file_object = open(file_name [, access_mode][, buffering])
```

```
file_object = open(file_name [, access_mode][, buffering])
```

- **file_name** – The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode** – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. This is optional parameter and the default file access mode is read (r).
  - r: Opens a file for reading only. The file pointer is placed at the beginning of the file.
  - w: Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

I/O in Python: Reading a file

- `fobj = open("filename", "r")`
  - The "r" is optional
- Close it when finished working with the file
- `fobj.close()`



access_mode:
Read: r
Write: w
Append:a

```
fobj = open("shakes.txt")
for line in fobj:
    print(line.rstrip())
fobj.close()
```

Open shakes.txt file
Read file object fobj one line at a time
Print each line after stripping the whitespaces
Close the file object

Reading a dictionary of passwords

```
#I/O in Python: Reading a file
fobj = open("dictionary.txt")
for line in fobj:
    print(line.strip())
fobj.close()
```

```
wfuzz
test
robots
about
test1
test2
admin
Admin
password
```

Activity: add line number with each line

```
# Reading a file

fobj = open("dictionary.txt")
for line in fobj:
    print(line.strip())
fobj.close()
```

```
#I/O in Python: Reading a file
fobj = open("dictionary.txt")
count=0
for line in fobj:
    count+=1
    print(count, line.strip())
fobj.close()
```

```
1 wfuzz
2 test
3 robots
4 about
5 test1
6 test2
7 admin
8 Admin
9 password
```

Reading in one go

Returns a list, convenient if the file is not too large

```
>>> text = open("dictionary.txt").readlines()
>>> print(text[:5])
```

Write into a File

```
fh = open("example.txt", "w")
fh.write("To write or not to write\nthat is the question!\n")
fh.close()
```

- If the file example.txt already exits, it will be overwritten
- To append an existing file use access mode “a”

```
fh = open("example.txt", "a")
fh.write("To write or not to write\nthat is the question!\n")
fh.close()
```

Activity: Reading and writing simultaneously

- Read “dictionary.txt” line by line, add line number and then write it in a new file.

CSV: Comma **S**eparated **V**alues

- CSV files are actually text files in which values are separated by comma
- CSV is a simple file format used to store tabular data, such as a spreadsheet or database.
- Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or OpenOffice Calc.

Reading records from a CSV file

SN	Name	City
1	Michael	New Jersey
2	Jack	California
3	Donald	Texas

The records are represented as a text file:

```
SN, Name, City
1, Michael, New Jersey
2, Jack, California
3, Donald, Texas
```

We can read it the same way as you read a text file

```
fobj = open("name-city.csv")
for line in fobj:
    print(line.strip())
fobj.close()
```

Processing the CSV further to extract a list of tuple

- We want the file to convert into a list of tuple, where each tuple represents a record:

```
[
  (SN, Name, City)
  (1, Michael, New Jersey),
  (2, Jack, California),
  (3, Donald, Texas)
]
```

First attempt

```
fobj = open("name-city.csv")
for line in fobj:
    rec = line.rstrip()
    values = rec.split(",")
    print(values)
fobj.close()
```

```
['SN', ' Name', ' City']
['1', ' Michael', ' New Jersey']
['2', ' Jack', ' California']
['3', ' Donald', ' Texas']
```

Processing the CSV further to extract a list of tuple

Final Attempt:

```
fobj = open("name-city.csv")
records = []
for line in fobj:
    rec = line.strip()
    values = rec.split(",")
    records.append(tuple(values))
fobj.close()

print(records)
```

Using CSV library to read

```
import csv

csvFile = open('name-city.csv')
reader = csv.reader(csvFile)
for row in reader:
    for cell in row:
        print(cell, end="\t")
    print()
```

```
SN      Name    City
1       Michael         New Jersey
2       Jack    California
3       Donald  Texas
```

Using CSV file to write data

```
import csv
csvData = [
    ['Name','Age'],
    ['Peter','22'],
    ['Sam','21']
  ]
csvFile = open('persons.csv','w')
writer = csv.writer(csvFile)
writer.writerows(csvData)
print("Data written successfully!")
```

```
Data written successfully!
```

```
import csv
csvFile = open('persons.csv','w')
writer = csv.writer(csvFile)
writer.writerow(['Name','Age'])
writer.writerow(['Peter','22'])
writer.writerow(['Sam','21'])

print("Data written successfully!")
```

```
Data written successfully!
```

كلّيات التقنية العليا
HIGHER COLLEGES OF TECHNOLOGY

CSF 2113
Programming for Information Security

Fundamentals of File Handling

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

Abu Dhabi Men's College

I/O in Python: Reading a file

- Python provides basic functions and methods necessary to manipulate files by default using a **file** object.
- The *open* Function: Before you can read or write a file, you have to open it using Python's built-in *open()* function.
  - This function creates a **file** object, which would be utilized to call other support methods associated with it.

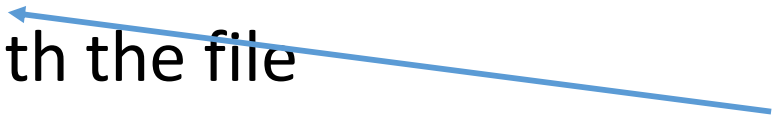
```
>>>file_object = open(file_name [, access_mode][, buffering])
```

```
file_object = open(file_name [, access_mode][, buffering])
```

- **file_name** – The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode** – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. This is optional parameter and the default file access mode is read (r).
  - r: Opens a file for reading only. The file pointer is placed at the beginning of the file.
  - w: Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

I/O in Python: Reading a file

- `fobj = open("filename", "r")`
  - The "r" is optional
- Close it when finished working with the file
- `fobj.close()`



access_mode:
Read: r
Write: w
Append:a

```
fobj = open("shakes.txt")
for line in fobj:
    print(line.rstrip())
fobj.close()
```

Open shakes.txt file
Read file object fobj one line at a time
Print each line after stripping the whitespaces
Close the file object

Reading a dictionary of passwords

```
#I/O in Python: Reading a file
fobj = open("dictionary.txt")
for line in fobj:
    print(line.strip())
fobj.close()
```

```
wfuzz
test
robots
about
test1
test2
admin
Admin
password
```

Activity: add line number with each line

```
# Reading a file

fobj = open("dictionary.txt")
for line in fobj:
    print(line.strip())
fobj.close()
```

```
#I/O in Python: Reading a file
fobj = open("dictionary.txt")
count=0
for line in fobj:
    count+=1
    print(count, line.strip())
fobj.close()
```

```
1 wfuzz
2 test
3 robots
4 about
5 test1
6 test2
7 admin
8 Admin
9 password
```

Reading in one go

Returns a list, convenient if the file is not too large

```
>>> text = open("dictionary.txt").readlines()
>>> print(text[:5])
```

Write into a File

```
fh = open("example.txt", "w")
fh.write("To write or not to write\nthat is the question!\n")
fh.close()
```

- If the file example.txt already exits, it will be overwritten
- To append an existing file use access mode “a”

```
fh = open("example.txt", "a")
fh.write("To write or not to write\nthat is the question!\n")
fh.close()
```

Activity: Reading and writing simultaneously

- Read “dictionary.txt” line by line, add line number and then write it in a new file.

CSV: Comma **S**eparated **V**alues

- CSV files are actually text files in which values are separated by comma
- CSV is a simple file format used to store tabular data, such as a spreadsheet or database.
- Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or OpenOffice Calc.

Reading records from a CSV file

SN	Name	City
1	Michael	New Jersey
2	Jack	California
3	Donald	Texas

The records are represented as a text file:

```
SN, Name, City
1, Michael, New Jersey
2, Jack, California
3, Donald, Texas
```

We can read it the same way as you read a text file

```
fobj = open("name-city.csv")
for line in fobj:
    print(line.strip())
fobj.close()
```

Processing the CSV further to extract a list of tuple

- We want the file to convert into a list of tuple, where each tuple represents a record:

```
[
  (SN, Name, City)
  (1, Michael, New Jersey),
  (2, Jack, California),
  (3, Donald, Texas)
]
```

First attempt

```
fobj = open("name-city.csv")
for line in fobj:
    rec = line.rstrip()
    values = rec.split(",")
    print(values)
fobj.close()
```

```
['SN', ' Name', ' City']
['1', ' Michael', ' New Jersey']
['2', ' Jack', ' California']
['3', ' Donald', ' Texas']
```

Processing the CSV further to extract a list of tuple

Final Attempt:

```
fobj = open("name-city.csv")
records = []
for line in fobj:
    rec = line.strip()
    values = rec.split(",")
    records.append(tuple(values))
fobj.close()

print(records)
```

Using CSV library to read

```
import csv

csvFile = open('name-city.csv')
reader = csv.reader(csvFile)
for row in reader:
    for cell in row:
        print(cell, end="\t")
    print()
```

```
SN      Name    City
1       Michael         New Jersey
2       Jack    California
3       Donald  Texas
```

Using CSV file to write data

```
import csv
csvData = [
    ['Name','Age'],
    ['Peter','22'],
    ['Sam','21']
  ]
csvFile = open('persons.csv','w')
writer = csv.writer(csvFile)
writer.writerows(csvData)
print("Data written successfully!")
```

```
Data written successfully!
```

```
import csv
csvFile = open('persons.csv','w')
writer = csv.writer(csvFile)
writer.writerow(['Name','Age'])
writer.writerow(['Peter','22'])
writer.writerow(['Sam','21'])

print("Data written successfully!")
```

```
Data written successfully!
```

كلّيـــات التقنيـــة العليـــا
HIGHER COLLEGES OF TECHNOLOGY

CSF 2113
Programming for Information Security

Web crawling with Scrapy (CLO4)

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

Abu Dhabi Men's College

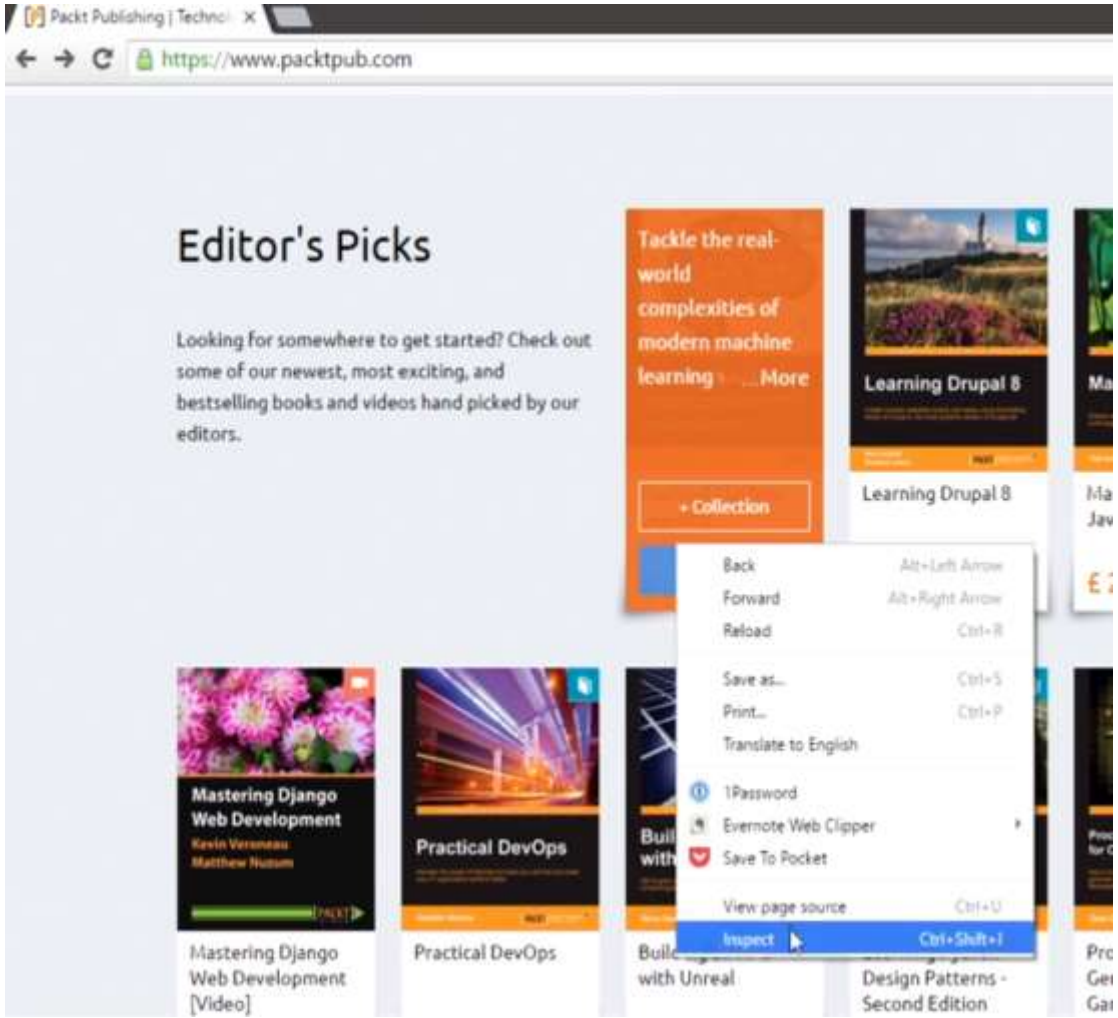
Table of Contents

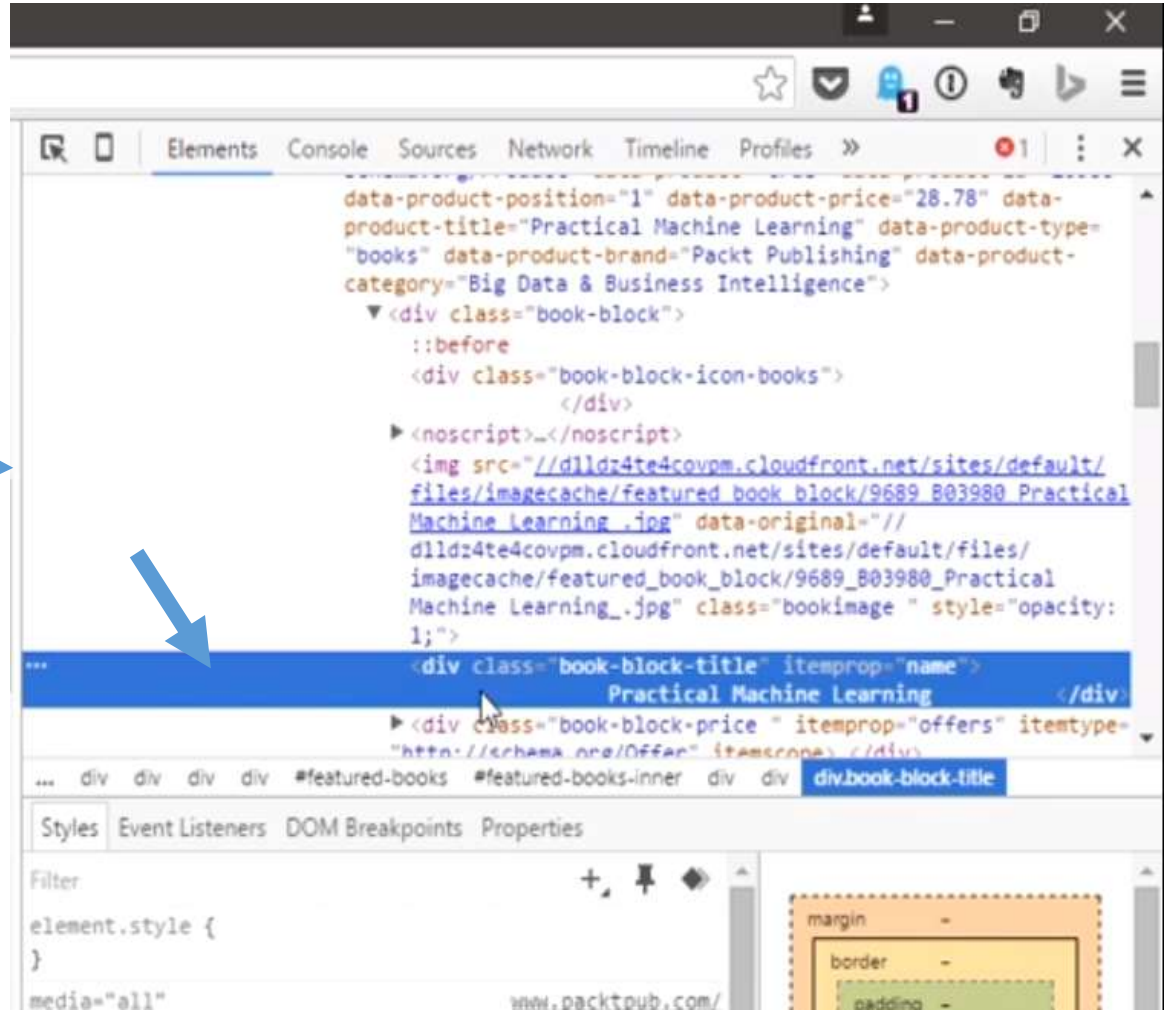
Creating a Web Crawler with Scrapy

- Decide what to scrap
- First Scrapy project
- Creating the spider for Scrapy
- Running the crawler

Decide what to scrap

- We need to define the objective (Extract all book titles form www.packtpub.com)
- Analyze the target: right click on a book title -> inspect





Scraping book titles

- We can see **`div`** with a class of **`book-block-title`** and then the title **`name`**.
- We need this to define what we want to extract in our crawl process

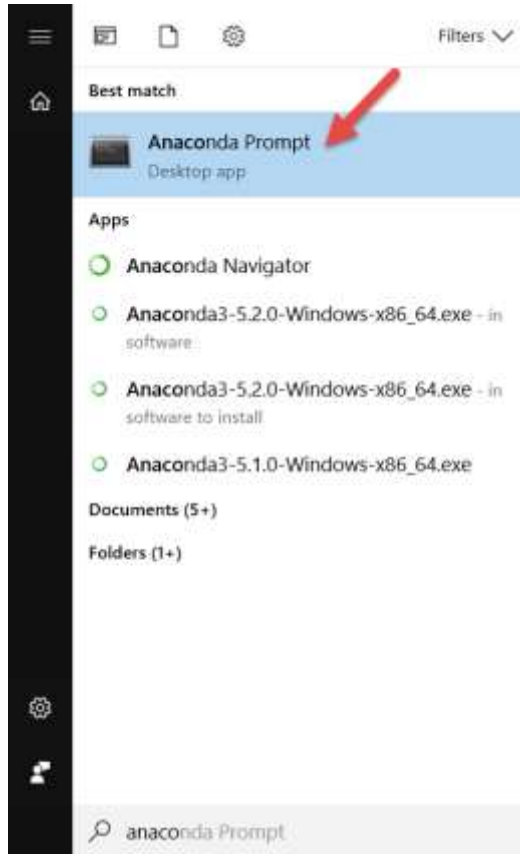
```
<div class="book-block-title" itemprop="name">
                    Practical Machine Learning    </div>
```

Creating first Scrapy project

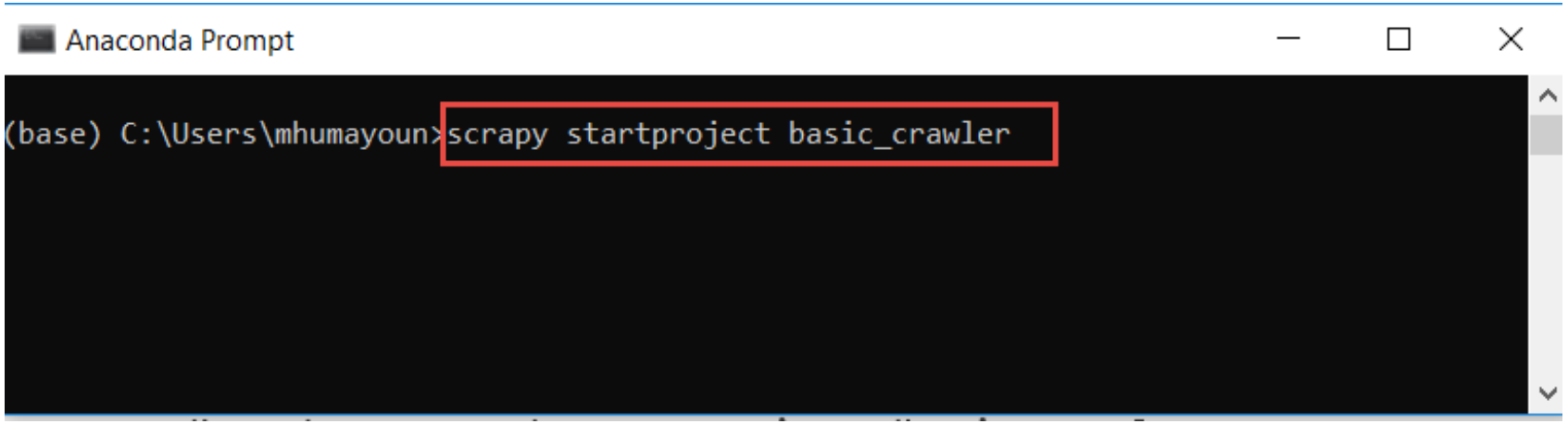
- Start a new Scrapy project:

```
$ scrapy startproject basic_crawler
```

```
New Scrapy project 'basic_crawler' created in:
    /home/pentester/scrapy_projects/basic_crawler
You can start your first spider with:
        cd basic_crawler
        scrapy genspider example example.com
```



Execute the command in 'Anaconda prompt' (for windows 10, in bash for Linux)

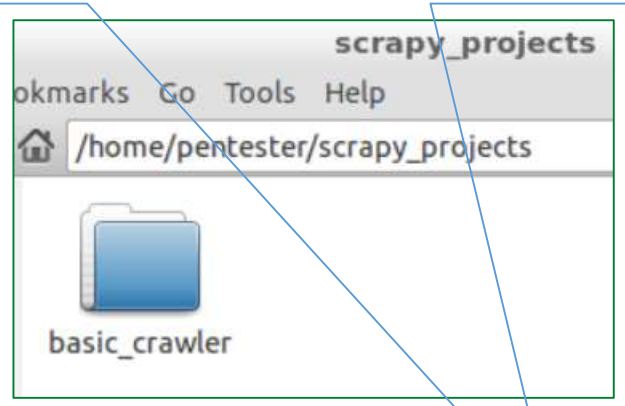


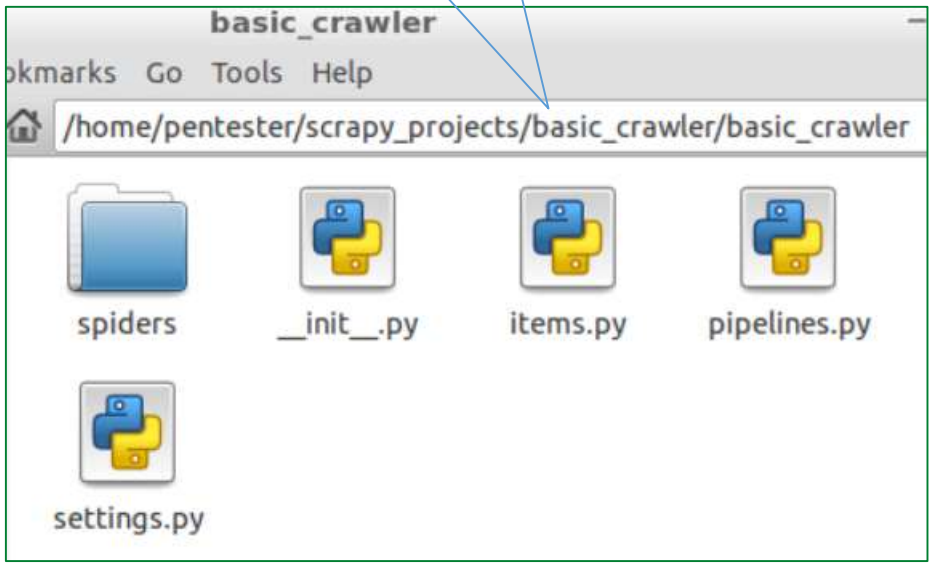
Creating first Scrapy project

Inside the **basic_crawler** directory, you will see another folder called basic_crawler

- Start a new Scrapy project:

```
$ scrapy startproject basic_crawler
```





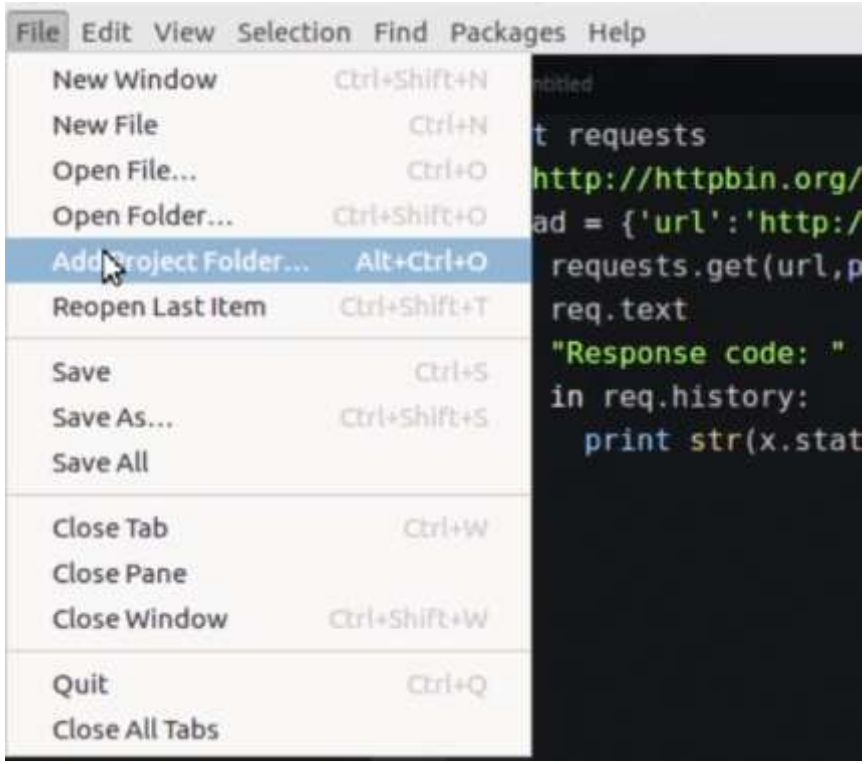

```
New Scrapy project 'basic_crawler' created in:
    /home/pentester/scrapy_projects/basic_crawler
You can start your first spider with:
        cd basic_crawler
        scrapy genspider example example.com
```

- When we create a project, Scrapy automatically generates a folder with the basic structure of the crawler

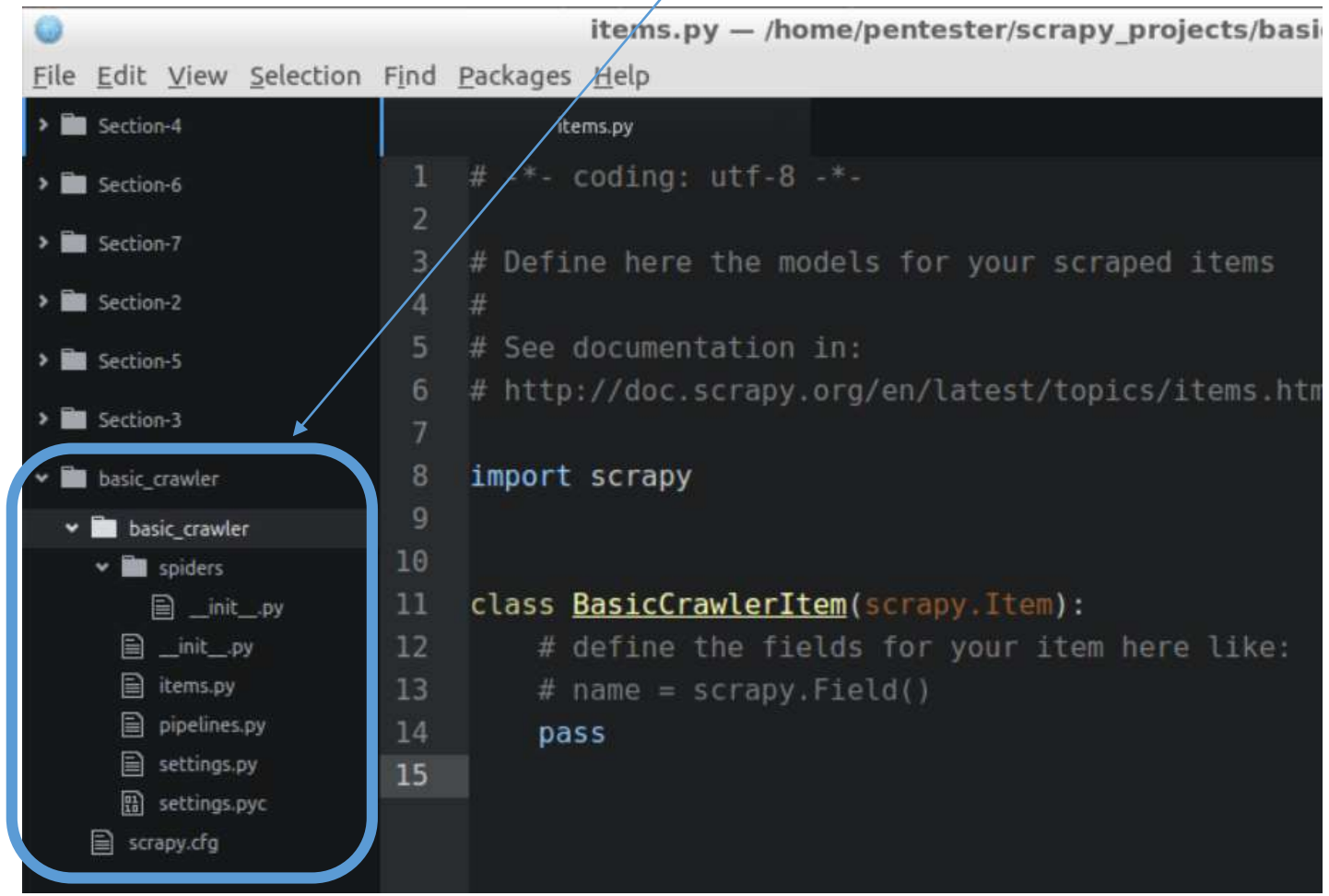
We shall work with the content of the `spiders` folder

First Scrapy project

- Open the Atom editor, add this project with ”add a project folder”, and select **`basic_crawler`**



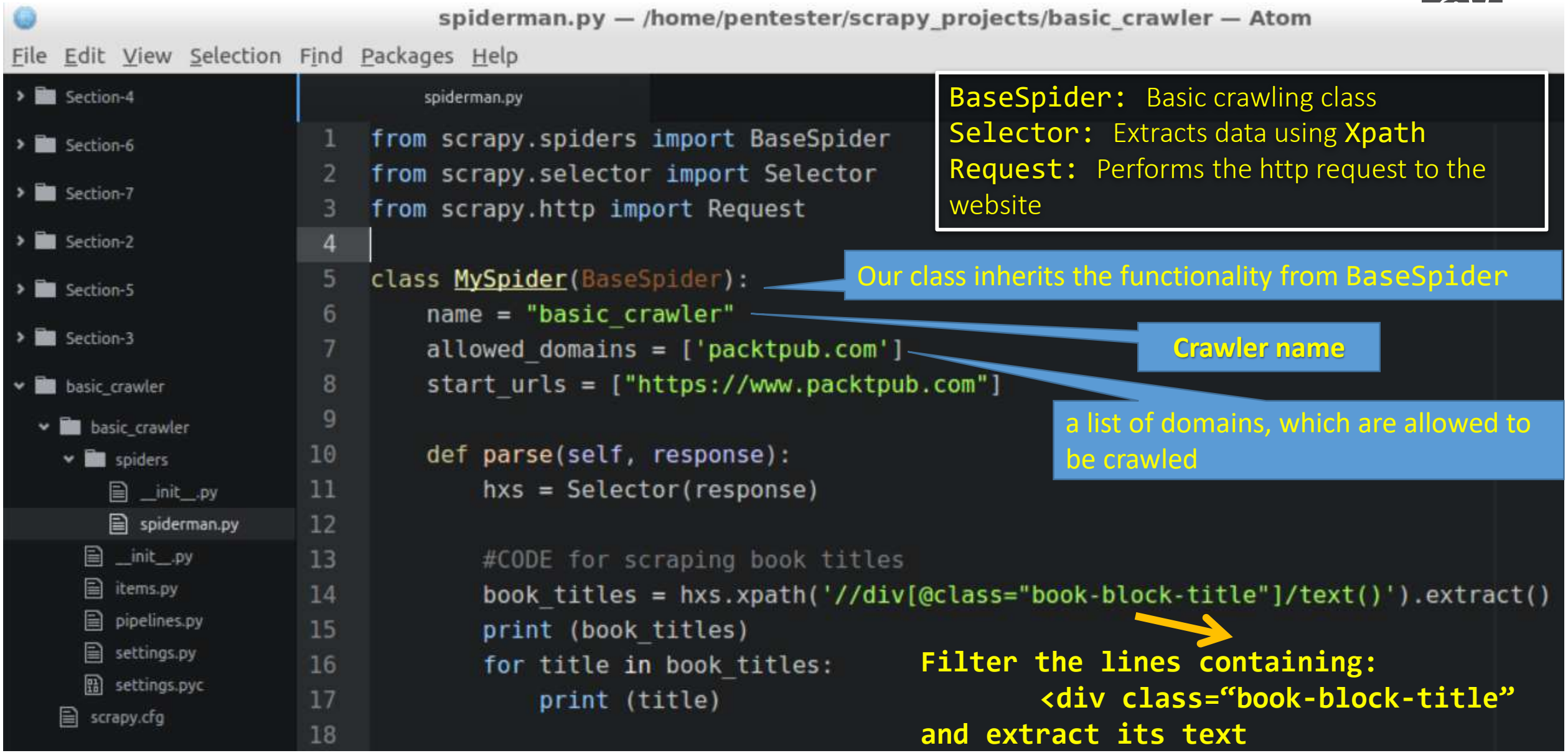
The basic structure of a Scrapy project



Directory structure of the project

```
basic_crawler/
   scrapy.cfg                          # deploy configuration file

   basic_crawler/                      # project's Python module, you'll import your code from here
            __init__.py

            items.py                   # project items definition file

            middlewares.py      # project middlewares file

            pipelines.py               # project pipelines file

            settings.py                # project settings file

            spiders/                   # a directory where you'll later put your spiders
                                                    # we'll create a spider in this folder
              __init__.py
```

Create new Python file spiderman.py in 'basic_crawler/spiders' directory



Scraping book titles

- We can see **`div`** with a class of **`book-block-title`** and then the title **`name`**.
- We need this to define what we want to extract in our crawl process

```
<div class="book-block-title" itemprop="name">
                    Practical Machine Learning    </div>
```

Corresponds to the xpath:

```
//div[@class="book-block-title"]/text()
```

Create new Python file `spiderman.py` in `'basic_crawler/spiders'` directory

- **`name`**: identifies the Spider. It must be unique within a project, that is, you can't set the same name for different Spiders.

```
class MySpider(BaseSpider):
    name = "basic_crawler"
```

- **`parse()`**: a method that will be called to handle the response downloaded for each of the requests made.
  - The response parameter is an instance of TextResponse that holds the page content and has further helpful methods to handle it.

```
def parse(self, response):
```

Executing the code

HCT 2.0

```
~/scrapy_projects/basic_crawler$ scrapy crawl basic_crawler
```

```
2018-07-03 13:26:04 [scrapy] DEBUG: Crawled (200) <GET https://www.packtpub.com> (referer: None)
[u'\n\t\t\t\tLearn Kotlin Programming [Video]\t\t\t', u'\n\t\t\t\tAngular 6 for Enterprise-Ready Web Applications\t\t\t', u'\n\t
tack for Architects - Second Edition\t\t\t', u'\n\t\t\t\tPractical DevOps - Second Edition\t\t\t', u'\n\t\t\t\tBeginning Swift\t
\t\t\t\tDocker Fundamentals [Integrated Course]\t\t\t', u'\n\t\t\t\tHands-On Machine Learning with C#\t\t\t', u'\n\t\t\t\tMaster
t', u'\n\t\t\t\tLearning TypeScript 2.x - Second Edition\t\t\t', u'\n\t\t\t\tJava 9: Building Robust Modular Applications\t\t\t'
t\tPython Machine Learning - Second Edition\t\t\t', u'\n\t\t\t\tDelphi High Performance\t\t\t', u'\n\t\t\t\tPython Web Scraping
t\t', u'\n\t\t\t\tFull Stack Development with JHipster\t\t\t', u'\n\t\t\t\tMastering Qlik Sense\t\t\t', u'\n\t\t\t\tPython Progr
prints\t\t\t', u'\n\t\t\t\tEssentials of Bitcoin and Blockchain [Video]\t\t\t', u'\n\t\t\t\tArchitecting Angular Applications wi
xJS, and NgRx\t\t\t', u'\n\t\t\t\tFull-Stack Vue.js 2 and Laravel 5\t\t\t', u'\n\t\t\t\tMastering Microsoft Power BI\t\t\t']
```

```
Learn Kotlin Programming [Video]

Angular 6 for Enterprise-Ready Web Applications

OpenStack for Architects - Second Edition

Practical DevOps - Second Edition

Beginning Swift

Docker Fundamentals [Integrated Course]
```

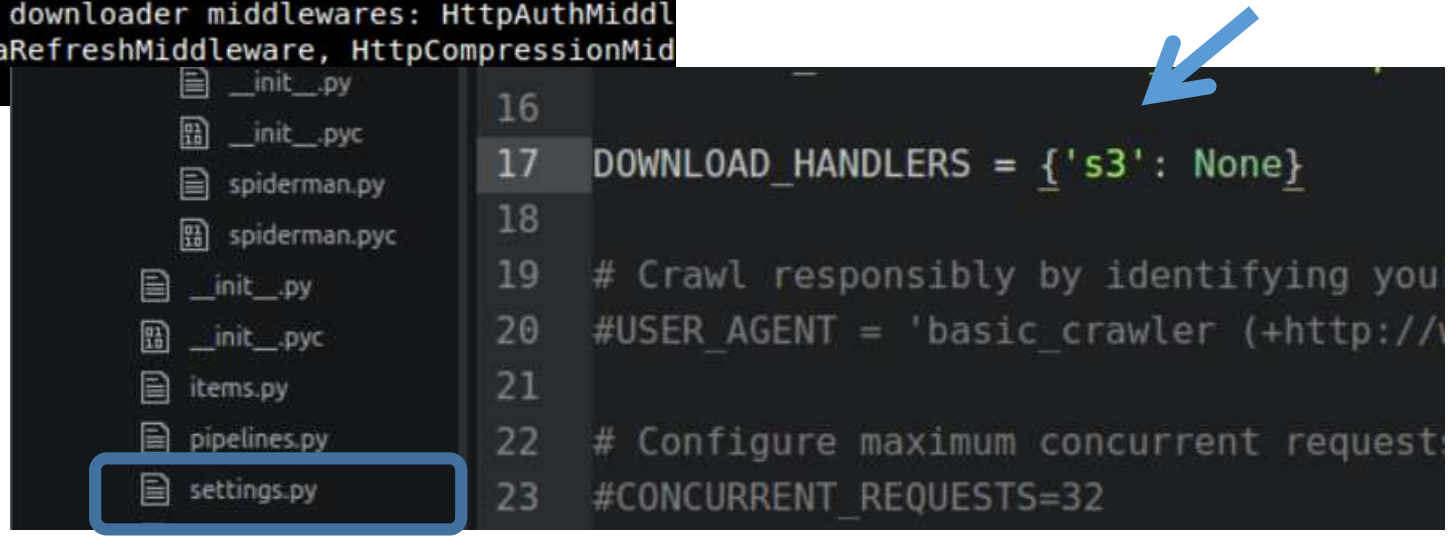
And so on

basic_crawler because it is the name of our spider i.e.spiderman.py

Attention: If you see the following error message:

```
2018-07-03 13:21:59 [boto] DEBUG: Retrieving credentials from metadata server.
2018-07-03 13:21:59 [boto] ERROR: Caught exception reading instance data
Traceback (most recent call last):
  File "/usr/lib/python2.7/dist-packages/boto/utils.py", line 210, in retry_url
    r = opener.open(req, timeout=timeout)
  File "/usr/lib/python2.7/urllib2.py", line 431, in open
    response = self._open(req, data)
  File "/usr/lib/python2.7/urllib2.py", line 449, in _open
    '_open', req)
  File "/usr/lib/python2.7/urllib2.py", line 409, in _call_chain
    result = func(*args)
  File "/usr/lib/python2.7/urllib2.py", line 1227, in http_open
    return self.do_open(httplib.HTTPConnection, req)
  File "/usr/lib/python2.7/urllib2.py", line 1197, in do_open
    raise URLError(err)
URLError: <urlopen error [Errno 101] Network is unreachable>
2018-07-03 13:21:59 [boto] ERROR: Unable to read instance data, giving up
2018-07-03 13:21:59 [scrapy] INFO: Enabled downloader middlewares: HttpAuthMiddl
yMiddleware, DefaultHeadersMiddleware, MetaRefreshMiddleware, HttpCompressionMid
sferMiddleware, DownloaderStats
```

It is a scrapy bug, either
upgrade to scrapy 1.x or add:
`DOWNLOAD_HANDLERS = {'s3': None}`
to `settings.py`



The First Proper Scrapy project

- Start a new Scrapy project:

```
$ scrapy startproject basic_crawler_proper
```

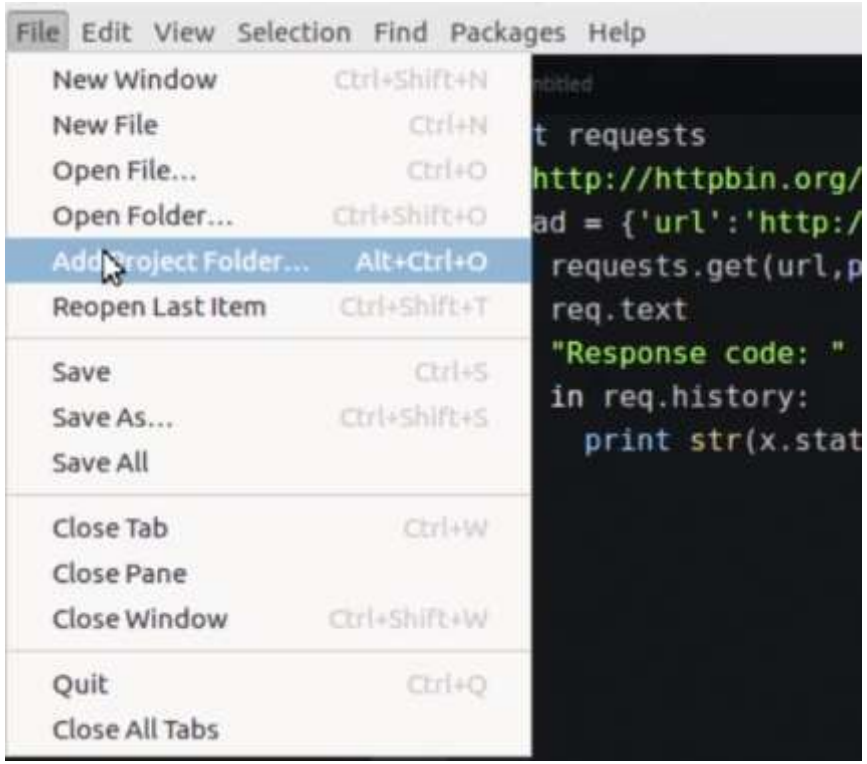
New Scrapy project 'basic_crawler_proper' created in:
/home/pentester/scrapy_projects/basic_crawler_proper

You can start your first spider with:
cd basic_crawler_proper
scrapy genspider example example.com

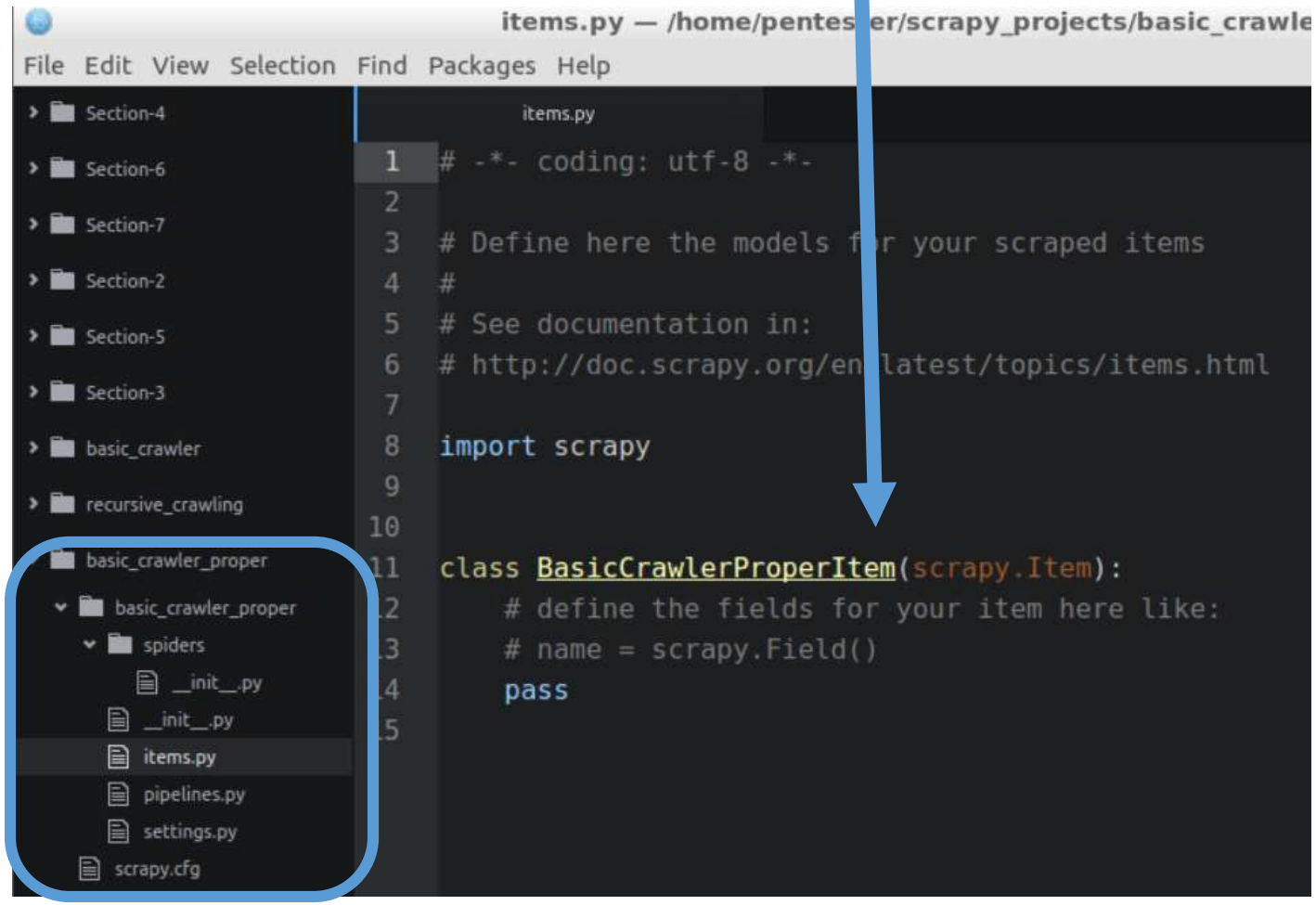
Execute in ‘Anaconda prompt’
(for windows 10, in bash for Linux)

The First Proper Scrapy project

- Open the Atom editor, add this project with ”add a project folder”, and select **basic_crawler_proper**



- We need to specify what are the things **we are interested in getting** while crawling a website
- These things are called **`items`** in Scrapy and think about them as **our data model**
- Edit the **`items.py`** file and define our first item
- We can see the class **`BasicCrawlerItem`** was created



Add this line and delete `pass` which was already written

Step1: Working on the class `BasicCrawlerItem` in `items.py`

```
import scrapy


class BasicCrawlerItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    title = scrapy.Field()
    |
```

Step2: Create new py file `spiderman.py` in '`basic_crawler/spiders`' directory as before

```
1  from scrapy.spiders import BaseSpider
2  from scrapy.selector import Selector
3  from scrapy.http import Request
4  from basic_crawler_proper.items import BasicCrawlerProperItem
5
6  class MySpider(BaseSpider):
7      name = "basic_crawler_proper"
8      allowed_domains = ['packtpub.com']
9      start_urls = ["https://www.packtpub.com"]
10
11     def parse(self, response):
12         hxs = Selector(response)
13
14         #CODE for scraping book titles
15         book_titles = hxs.xpath('//div[@class="book-block-title"]/text()').extract()
16         for title in book_titles:
17             book = BasicCrawlerProperItem()
18             book["title"] = title
19             yield book
```

Executing the code

```
~/scrapy_projects/basic_crawler_proper$ scrapy crawl basic_crawler_proper
```

```
{'title': u'\n\t\t\t\tLearn Kotlin Programming [Video]\t\t\t'}
2018-07-03 15:13:00 [scrapy] DEBUG: Scraped from <200 https://www.packtpub.com>
{'title': u'\n\t\t\t\tAngular 6 for Enterprise-Ready Web Applications\t\t\t'}
2018-07-03 15:13:00 [scrapy] DEBUG: Scraped from <200 https://www.packtpub.com>
{'title': u'\n\t\t\t\tOpenStack for Architects - Second Edition\t\t\t'}
2018-07-03 15:13:00 [scrapy] DEBUG: Scraped from <200 https://www.packtpub.com>
{'title': u'\n\t\t\t\tPractical DevOps - Second Edition\t\t\t'}
2018-07-03 15:13:00 [scrapy] DEBUG: Scraped from <200 https://www.packtpub.com>
{'title': u'\n\t\t\t\tBeginning Swift\t\t\t'}
2018-07-03 15:13:00 [scrapy] DEBUG: Scraped from <200 https://www.packtpub.com>
{'title': u'\n\t\t\t\tDocker Fundamentals [Integrated Course]\t\t\t'}
2018-07-03 15:13:00 [scrapy] DEBUG: Scraped from <200 https://www.packtpub.com>
{'title': u'\n\t\t\t\tHands-On Machine Learning with C#\t\t\t'}
```

Writing in a file in json format (must be run from `~/scrapy_projects/basic_crawler_proper$`):

```
$ scrapy crawl basic_crawler_proper -o books.json –t json
```

books.json

File Edit Search Options Help

```
[{"title": "\n\t\t\t\tLearn Kotlin Programming [Video]\t\t\t"},
{"title": "\n\t\t\t\tAngular 6 for Enterprise-Ready Web Applications\t\t\t"}
{"title": "\n\t\t\t\tOpenStack for Architects - Second Edition\t\t\t"},
{"title": "\n\t\t\t\tPractical DevOps - Second Edition\t\t\t"},
{"title": "\n\t\t\t\tBeginning Swift\t\t\t"},
{"title": "\n\t\t\t\tDocker Fundamentals [Integrated Course]\t\t\t"},
{"title": "\n\t\t\t\tHands-On Machine Learning with C#\t\t\t"},
{"title": "\n\t\t\t\tMastering Go\t\t\t"},
{"title": "\n\t\t\t\tLearning TypeScript 2.x - Second Edition\t\t\t"},
{"title": "\n\t\t\t\tJava 9: Building Robust Modular Applications\t\t\t"},
```

Writing in a file in csv format:

```
$ scrapy crawl basic_crawler_proper
-o books.csv –t csv
```

JSON: **J**ava**S**cript **O**bject **N**otation

- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.
- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

JSON examples:


```
'{"name":"John", "age":31, "city":"New York"}';
[{“Title”:“\n\tLearn Kotlin programming…”},
{{“Title”:“\n\tAngular 6 for Enterprise-Ready Web…”},
      …
]
```

books.json

File Edit Search Options Help

[{"title": "\n\t\t\t\tLearn Kotlin Programming [Video]\t\t\t"},
{"title": "\n\t\t\t\tAngular 6 for Enterprise-Ready Web Applications\t\t\t"}
{"title": "\n\t\t\t\tOpenStack for Architects - Second Edition\t\t\t"},
{"title": "\n\t\t\t\tPractical DevOps - Second Edition\t\t\t"},
{"title": "\n\t\t\t\tBeginning Swift\t\t\t"},
{"title": "\n\t\t\t\tDocker Fundamentals [Integrated Course]\t\t\t"},
{"title": "\n\t\t\t\tHands-On Machine Learning with C#\t\t\t"},
{"title": "\n\t\t\t\tMastering Go\t\t\t"},
{"title": "\n\t\t\t\tLearning TypeScript 2.x - Second Edition\t\t\t"},
{"title": "\n\t\t\t\tJava 9: Building Robust Modular Applications\t\t\t"},

CSV: Comma **S**eparated **V**alues

- CSV is a simple file format used to store tabular data, such as a spreadsheet or database.
- Files in the CSV format can be imported to and exported from programs that **store data in tables**, such as Microsoft Excel or OpenOffice Calc.

Name	Class	Dorm	Room	GPA
Sally Whittaker	2018	McCarren House	312	3.75
Belinda Jameson	2017	Cushing House	148	3.52
Jeff Smith	2018	Prescott House	17-D	3.20
Sandy Allen	2019	Oliver House	108	3.48

```
Sally Whittaker,2018,McCarren House,312,3.75
Belinda Jameson,2017,Cushing House,148,3.52
Jeff Smith,2018,Prescott House,17-D,3.20
Sandy Allen,2019,Oliver House,108,3.48
```

Know more about xpaths

- http://pythonscraping.com/blog/xpath-and-scrapy
- https://blog.michaelyin.info/scrapy-tutorial-7-how-use-xpath-scrapy/

كلّيات التقنية العليا
HIGHER COLLEGES OF TECHNOLOGY

CSF 2113
Programming for Information Security

Penetration Testing (CLO 05)

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

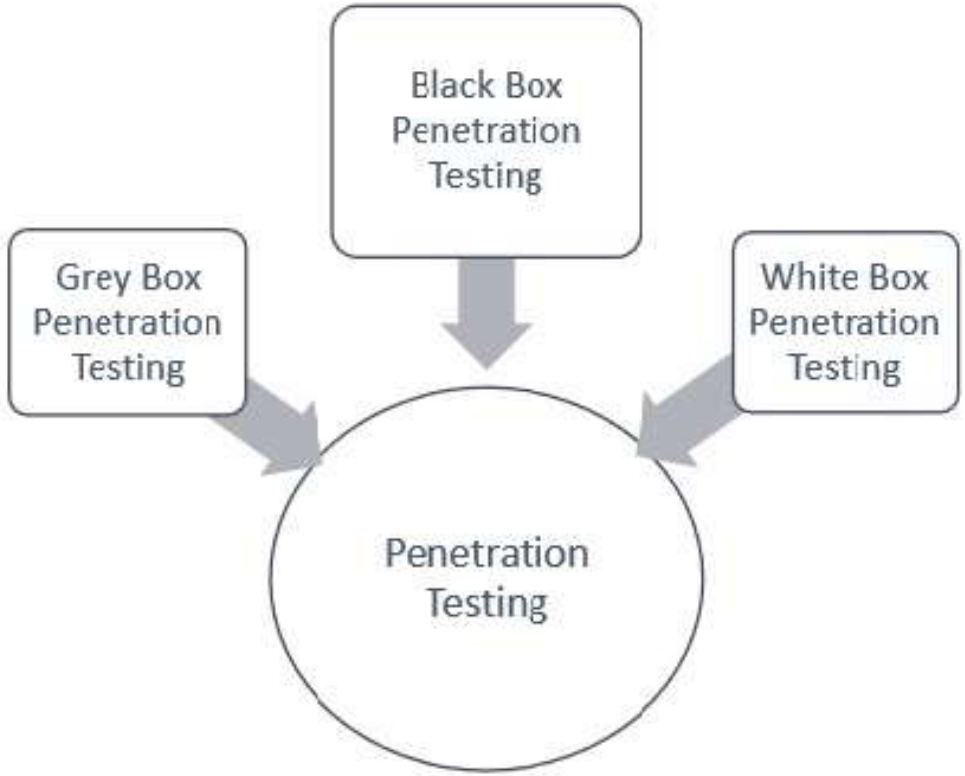
Abu Dhabi Men's College

Introduction

- Penetration Testing is used to find flaws in the system in order to take appropriate security measures to **protect** the data and maintain functionality.

Types of Pen Testing

- Black Box Penetration Testing
- White Box Penetration Testing
- Grey Box Penetration Testing



Black Box Penetration Testing

- Tester has no idea about the systems that he is going to test.
- In other words, the Penn tester is given “ZERO” information about the system inner workings
- He is interested to gather information about the target network or system. For example,
  - in this testing, a tester only knows what should be the expected outcome and he does not know how the outcomes arrives
  - He does not examine any programming codes

Advantages of Black Box Penetration Testing

- Tester need not necessarily be an expert, as it does not demand specific language knowledge
- Tester verifies contradictions in the actual system and the specifications
- Test is generally conducted with the perspective of a user, not the designer

Disadvantages of Black Box Penetration Testing

- Particularly, these kinds of test cases are difficult to design.
- Possibly, it is not worth, incase designer has already conducted a test case.
- It does not conduct everything.

White Box Penetration Testing

- This is a comprehensive testing, as tester has been provided with whole range of information about the systems and/or network such as Schema, Source code, OS details, IP address, etc.
- White box penetration testing examines the code coverage and does data flow testing, path testing, loop testing, etc.

Advantages of White Box Penetration Testing

- It ensures that all independent paths of a module have been exercised.
- It ensures that all logical decisions have been verified along with their true and false value.
- It discovers the logical errors
- It finds the design errors that may have occurred because of the difference between logical flow of the program and the actual execution

Disadvantages of White Box Penetration Testing

- A wide range of expertise is needed to perform such testing
- As a consequence, it is expensive

Grey Box Penetration Testing

- A tester usually provides partial or limited information about the internal details of the program of a system
- Usually access of the source code is not provided to the tester
- It can be considered as an attack by an external hacker who had gained illegitimate access to an organization's network infrastructure documents.

Advantages of Grey Box Penetration Testing

- As there is clear difference between a developer and a tester, so there is least risk of personal conflict and bias
- You don’t need to provide the internal information about the program functions and other operations

Areas of Penetration Testing

- **Network Penetration Testing**
  - the physical structure of a system needs to be tested to identify the vulnerability and risk which ensures the security in a network.
- **Application Penetration Testing**
  - the logical structure of the system needs to be tested.
- **The response or workflow of the system**
  - test the ability of the respective organization to prevent unauthorized access to its information systems.
  - For instance, using social engineering that gathers information on human interaction

كلّيات التقنية العليا
HIGHER COLLEGES OF TECHNOLOGY

CSF 2113
Programming for Information Security

Resource Discovery (CLO 04)

Muhammad Humayoun, PhD

Assistant Professor

mhumayoun@hct.ac.ae

Abu Dhabi Men's College

This module is based on the course: **Learning Python Web Penetration Testing**, www.lynda.com

Table of Contents

What is Fuzzing?

- Fuzz testing (fuzzing) is a **Black Box software testing technique** used to discover coding errors and security loopholes in software, operating systems or networks.
- It involves inputting massive amounts of random data, called fuzz, to the test subject in an attempt to make it crash.
- If a vulnerability is found, a software tool called a fuzzer can be used to identify potential causes.
- Simple, but **offers a high benefit-to-cost ratio** and can often reveal serious defects that are overlooked when software is written and debugged.

What makes Fuzzing possible?

- User errors
- Fuzzing is only useful against a system which:
  - improperly sanitize input
  - take more data than it can handle
- **Sanitizing input:** checking user **input** before storing it in a database or using it for any other purpose to prevent malicious code injection.

Three important Fuzzing tasks that we'll cover:

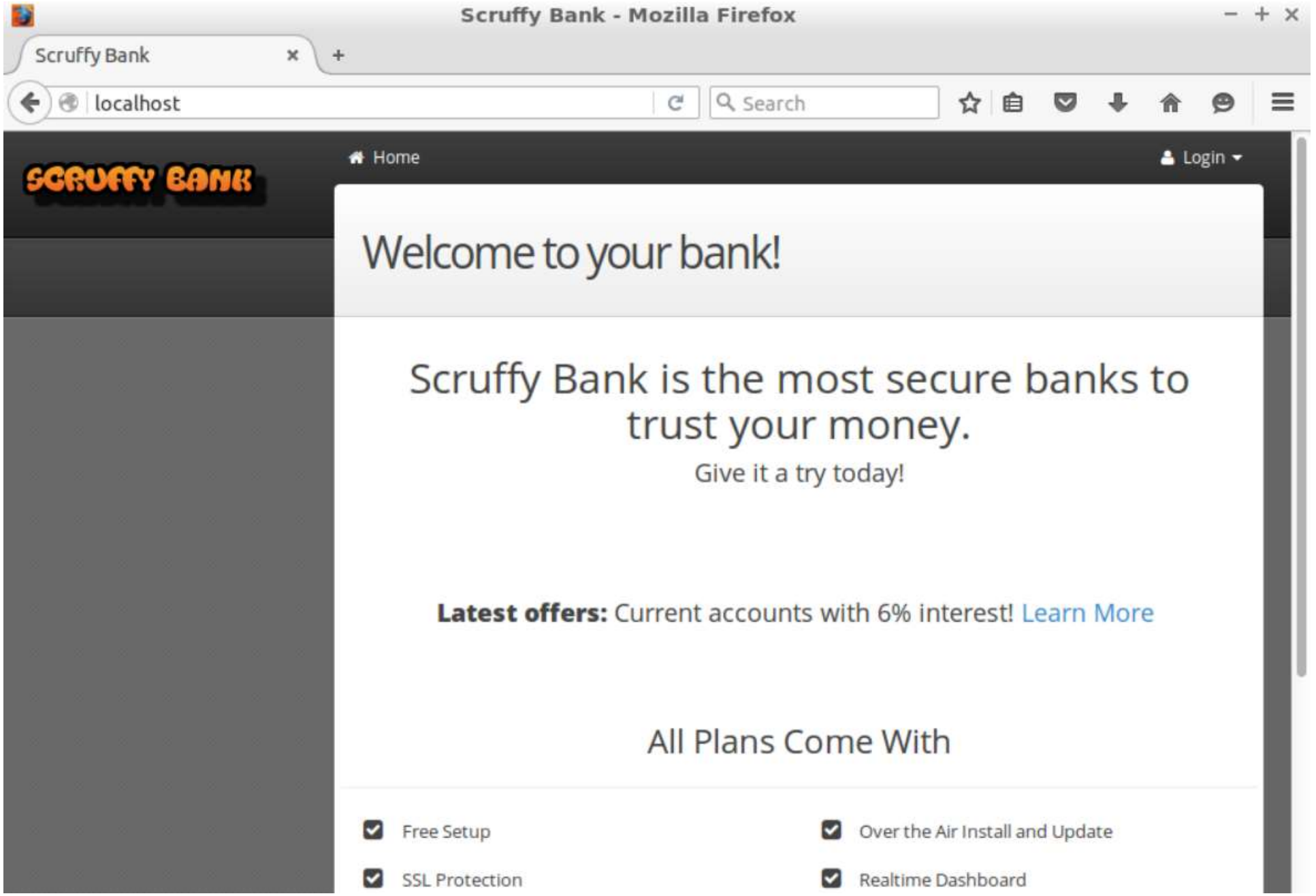
- Resource Discovery Week 12-13
- Password Cracking Week 12-13
- SQL Injections Week 14-15

What is Resource Discovery

- Discovering what resources are available in the application
- A process of building a map or catalog of the application pages and functionalities
- How: Dictionary attacks, Brute forcing

Building the first brute forcer for resource discovery using a dictionary of words

- Dictionary of words
  - The most commonly used resources in web applications. These dictionaries are plenty (we'll use `common.txt, commons.txt`)
- Define the objective of the tool
  - Make web links using dictionary http://somewebsite/admin (word admin comes from the dictionary)
  - Find out which pages, files, directories are found using http statuscode 200
- Run it against our test web application
  - The scruffy bank application running at **`localhost (127.0.0.1)`**


Scruffy Bank - Mozilla Firefox
Scruffy Bank
localhost
Search
SCRUFFY BANK
Home
Login
Welcome to your bank!
Scruffy Bank is the most secure banks to trust your money.
Give it a try today!
Latest offers: Current accounts with 6% interest! Learn More
All Plans Come With
Free Setup
Over the Air Install and Update
SSL Protection
Realtime Dashboard

Step 1: bruteforce-1.0.py – building the basic command line structure

bruteforce-1.0.py

```
1  import requests
2  import sys
3
4  def banner():
5      print "\n****************************************"
6      print "* BruteForcer for Resource Discovery 0.1 *"
7      print "****************************************"
8
9  def usage():
10     print "Usage:"
11     print "python bruteforce-1.0.py targetsite dictionary_file\n"
12     print "example: python bruteforce-1.0.py http://127.0.0.1 common.txt\n"
13
14 if __name__ == "__main__":
15     banner()
16     if len(sys.argv) != 3:
17             usage()
18             sys.exit()
19
```

Request library from web request
sys library from getting command line arguments

Defining the main function of our application

If arguments passed on the console are not 3
Print usage function and exit

```
20      args = sys.argv
21      #print args
22      url=args[1]+"/!!!!"
23      dict_f=args[2]
24
25      try:
26              f = open(dict_f, "r")
27              words = f.readlines()
28      except:
29              print ("Failed opening file: ")
30              print (str(dict_f)+"\n")
31              sys.exit()
32
33      #printing
34      print "Printing args:"
35      print args
36
37      print "printing url"
38      print url
39
40      print "printing words from the file"
41      print words[:10]
```

Get command line arguments

Add /!!!! to the url why?
We'll replace !!!! with dictionary words later

Printing first ten dictionary words for debugging

Step 1: bruteforce-1.0.py – Output

```
pentester@pentester-packt:~/resource_discovery$ python bruteforce-1.0.py http://127.0.0.1
common.txt

****************************************
* BruteForcer for Resource Discovery 0.1 *
****************************************
Printing args:
['bruteforce-1.0.py', 'http://127.0.0.1', 'common.txt']
printing url
http://127.0.0.1/!!!!
printing words from the file
['wfuzz\n', 'test\n', 'robots.txt\n', 'about.php\n', 'redir.php\n', 'test1.txt\n', 'test2.
txt\n', 'admin\n', 'Admin\n', 'index.php\n']
pentester@pentester-packt:~/resource_discovery$
```

words is a list of words
Need to remove \n

Step 2: bruteforce-1.1.py – Read words from dictionary one by one, make links and get those pages using Request library

HCT 2.0

```
26      try:
27              f = open(dict_f, "r")
28              words = f.readlines()
29      except:
30              print ("Failed opening file: ")
31              print (str(dict_f)+"\n")
32              sys.exit()
33
34      for w in words:
35          #print w
36          word = w.split("\n")[0]
37          urly = url.replace('!!!!',word)
38          #print (word + "    " + urly)
39          r = requests.get(urly)
40          print str(r.status_code) + " - " + urly
41
```

words is a list of words
Need to remove \n

w is a single word
Need to remove \n

`w.split("\n")` returns a list with two elems: `["word", ""]` get val at 0

url contains: `http://127.0.0.1/!!!!`
`Replace !!!! With word`
`e.g. http://127.0.0.1/wfuzz`
`2nd time: http://127.0.0.1/test`
`And so on …`

Get the page
`r.status_code` contains its status code

Step 2: bruteforce-1.1.py – Output


```
pentester@pentester-packt:~/resource_discovery$ python bruteforce-1.1.py http://127.0.0.1
common.txt

****************************************
* BruteForcer for Resource Discovery 0.1 *
****************************************
404 - http://127.0.0.1/wfuzz
404 - http://127.0.0.1/test
200 - http://127.0.0.1/robots.txt
404 - http://127.0.0.1/about.php
200 - http://127.0.0.1/redir.php
200 - http://127.0.0.1/test1.txt
200 - http://127.0.0.1/test2.txt
404 - http://127.0.0.1/admin
401 - http://127.0.0.1/Admin
200 - http://127.0.0.1/index.php
pentester@pentester-packt:~/resource_discovery$
```

- Try `commons.txt` instead of `common.txt`
- `commons.txt` is a very large file. There will be a lot more results with status code 404 (Page not found)
- **We are only interested in those links for which a page is found**
- So it seems better **not to print those links with 404 status code** for readability

Step 3: bruteforce-1.2.py – Do not print those results that have status code 404

HCT 2.0

```
26    try:
27            f = open(dict_f, "r")
28            words = f.readlines()
29    except:
30            print ("Failed opening file: ")
31            print (str(dict_f)+"\n")
32            sys.exit()
33
34    print "-----------------------------------------------------"
35    print "Code\t\tURL"
36    print "-----------------------------------------------------"
37
38    for w in words:
39        #print w
40        word = w.split("\n")[0]
41        urly = url.replace('!!!!',word)
42        #print (word + "    " + urly)
43        r = requests.get(urly)
44
45        #filter out 404 code (Page not found)
46        if (r.status_code==404):
47            pass
48        else:
49            print str(r.status_code) + "\t\t" +urly
```

Step 3: bruteforce-1.2.py – Output

```
pentester@pentester-packt:~/resource_discovery$ python bruteforce-1.2.py http://127.0.0.1 common.txt

****************************************
* BruteForcer for Resource Discovery 0.1 *
****************************************
------------------------------------------------
Code            URL
------------------------------------------------
200             http://127.0.0.1/robots.txt
200             http://127.0.0.1/redir.php
200             http://127.0.0.1/test1.txt
200             http://127.0.0.1/test2.txt
401             http://127.0.0.1/Admin
200             http://127.0.0.1/index.php
pentester@pentester-packt:~/resource_discovery$
```

```
pentester@pentester-packt:~/resource_discovery$ python bruteforce-1.2.py http://127.0.0.1 commons.txt

****************************************
* BruteForcer for Resource Discovery 0.1 *
****************************************
------------------------------------------------
Code            URL
------------------------------------------------
401             http://127.0.0.1/Admin
401             http://127.0.0.1/backoffice
200             http://127.0.0.1/css
200             http://127.0.0.1/img
200             http://127.0.0.1/js
200             http://127.0.0.1/phpmyadmin
200             http://127.0.0.1/ramon
200             http://127.0.0.1/robots.txt
200             http://127.0.0.1/test1.txt
200             http://127.0.0.1/test2.txt
200             http://127.0.0.1/redir.php
pentester@pentester-packt:~/resource_discovery$
```

Redirection page. Lets us visit it in browser

URL redirection

- URL redirection (also called URL forwarding), is a technique for making a web page available under more than one URL address.
- When a web browser attempts to open a URL that has been redirected, a page with a different URL is opened.
- URL redirection may be needed for various reasons:
  - for URL shortening
  - to prevent broken links when web pages are moved
  - to allow multiple domain names belonging to the same company (or organization) to refer to a single web site.
    - Ex. wikipedia.com and wikipedia.net are automatically redirected to wikipedia.org
  - for hostile purposes such as phishing attacks or malware distribution

URL redirection and resource discovery

- For discovering resources, detecting URL redirection is important
- Instead of saving redirect links, we want to save those urls where the redirects were taking us.

Redirect status codes and characteristics

HTTP Status Code	HTTP Version	Temporary / Permanent
301	HTTP/1.0	Permanent: The page has moved permanently
302	HTTP/1.0	Temporary: The page has moved temporarily
303	HTTP/1.1	Temporary: The page has moved temporarily
307	HTTP/1.1	Temporary: The page has moved temporarily
308	HTTP/1.1	Permanent: The page has moved permanently

Step 4: bruteforce-1.3.py – Detecting Redirections

HCT 2.0

```
39      for w in words:
40          #print w
41          word = w.split("\n")[0]
42          urly = url.replace('!!!!',word)+".php"
43          #print (word + "    " + urly)
44          r = requests.get(urly)
45          code = r.status_code
46
47          # Detecting redirections
48          if r.history != []:
49              first = r.history[0]
50              code = first.status_code
51              #print str(code)
52          else:
53              pass
54
55          #filter out 404 code (Page not found)
56          if (code==404):
57              pass
58          else:
59              print str(code) + "\t\t" +urly
```

Step 4: bruteforce-1.3.py – output

HCT 2.0

```
pentester@pentester-packt:~/resource_discovery$ python bruteforce-1.3.py http://127.0.0.1 commons.txt

*********************************************
* BruteForcer for Resource Discovery 0.1 *
*********************************************
-------------------------------------------------
Code            URL
-------------------------------------------------
401             http://127.0.0.1/Admin
401             http://127.0.0.1/backoffice
301             http://127.0.0.1/css
301             http://127.0.0.1/img
301             http://127.0.0.1/js
301             http://127.0.0.1/phpmyadmin
301             http://127.0.0.1/ramon
200             http://127.0.0.1/robots.txt
200             http://127.0.0.1/test1.txt
200             http://127.0.0.1/test2.txt
301             http://127.0.0.1/redir.php
pentester@pentester-packt:~/resource_discovery$
```

Step 5: bruteforce-1.4.py – Taking snapshots

HCT 2.0

First a simple example:

test-selenium.py

```
1  import requests
2  import time
3
4  from selenium import webdriver
5  from selenium.webdriver.common.keys import Keys
6  from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
7
8  url = "http://www.bing.com"
9  urly = "www.bing.com"
10
11 driver = webdriver.PhantomJS()
12 time.sleep(1)
13 driver.get(url)
14 driver.save_screenshot(urly +".png")
15 driver.quit()
16
```

Step 5: bruteforce-1.4.py – Taking snapshots

- Taking snapshot of every resource returning 200 status code
- A way to speed up the analysis of big apps or when testing multiple apps in a short period of time
- Needed libraries are:
  - **Selenium web driver** for Python and Phantom JS
    - Selenium web driver is a tool used to automate web browsers programmatically
  - **PhantomJS** is a headless browser

https://www.seleniumhq.org/

http://phantomjs.org/

Step 5: bruteforce-1.4.py – Taking snapshots

A test example

test-selenium.py

```
1  import requests
2  import time
3
4  from selenium import webdriver
5  from selenium.webdriver.common.keys import Keys
6  from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
7
8  url = "http://www.hct.ac.ae"
9  urly = "www.hct.ac.ae"
10
11 driver = webdriver.PhantomJS()
12 driver.set_window_size(1024, 1024)
13 time.sleep(1)
14 driver.get(url)
15 driver.save_screenshot(urly +".png")
16 driver.quit()
```

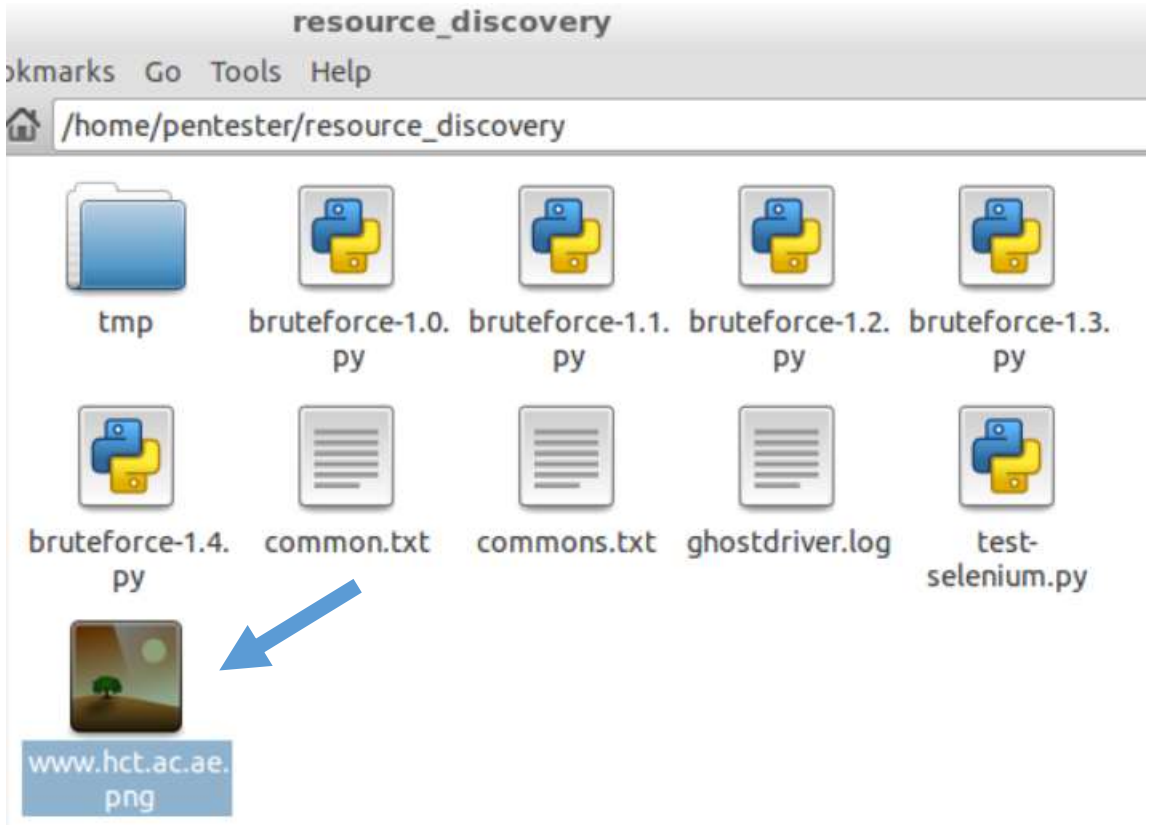
Try this instead: driver = webdriver.Firefox()

Step 5: bruteforce-1.4.py – Taking snapshots

HCT 2.0

A test example

```
$ python test-selenium.py
```



Step 5: bruteforce-1.4.py – Taking snapshots

HCT 2.0

bruteforce-1.4.py

```
1  import requests
2  import sys
3  import time
4  import os
5
6  from selenium import webdriver
7  from selenium.webdriver.common.keys import Keys
8  from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
9
10 def banner():
```

```
60          # create tmp directory if not exits
61          directory = "tmp"
62          if not os.path.exists(directory):
63              os.makedirs(directory)
64
```

Step 5: bruteforce-1.4.py – Taking snapshots

HCT 2.0

```
65          #filter out 404 code (Page not found)
66          if (code==404):
67              pass
68          else:
69              driver = webdriver.PhantomJS()
70              #driver = webdriver.Firefox()
71              time.sleep(1)
72              driver.set_window_size(1024, 768)
73              driver.get(urly)
74              driver.save_screenshot(directory+"/"+urly.split("/")[-1] +".png")
75              driver.close()
76              print str(code) + "\t\t" +urly
```

Step 5: bruteforce-1.4.py – Output


```
pentester@pentester-packt:~/resource_discovery$ python bruteforce-1.4.py http://
127.0.0.1 commons.txt

*****************************************
* BruteForcer for Resource Discovery 0.1 *
*****************************************
------------------------------------------------
Code            URL
------------------------------------------------
401             http://127.0.0.1/Admin
401             http://127.0.0.1/backoffice
200             http://127.0.0.1/css
200             http://127.0.0.1/img
200             http://127.0.0.1/js
200             http://127.0.0.1/phpmyadmin
200             http://127.0.0.1/ramon
200             http://127.0.0.1/robots.txt
200             http://127.0.0.1/test1.txt
200             http://127.0.0.1/test2.txt
```