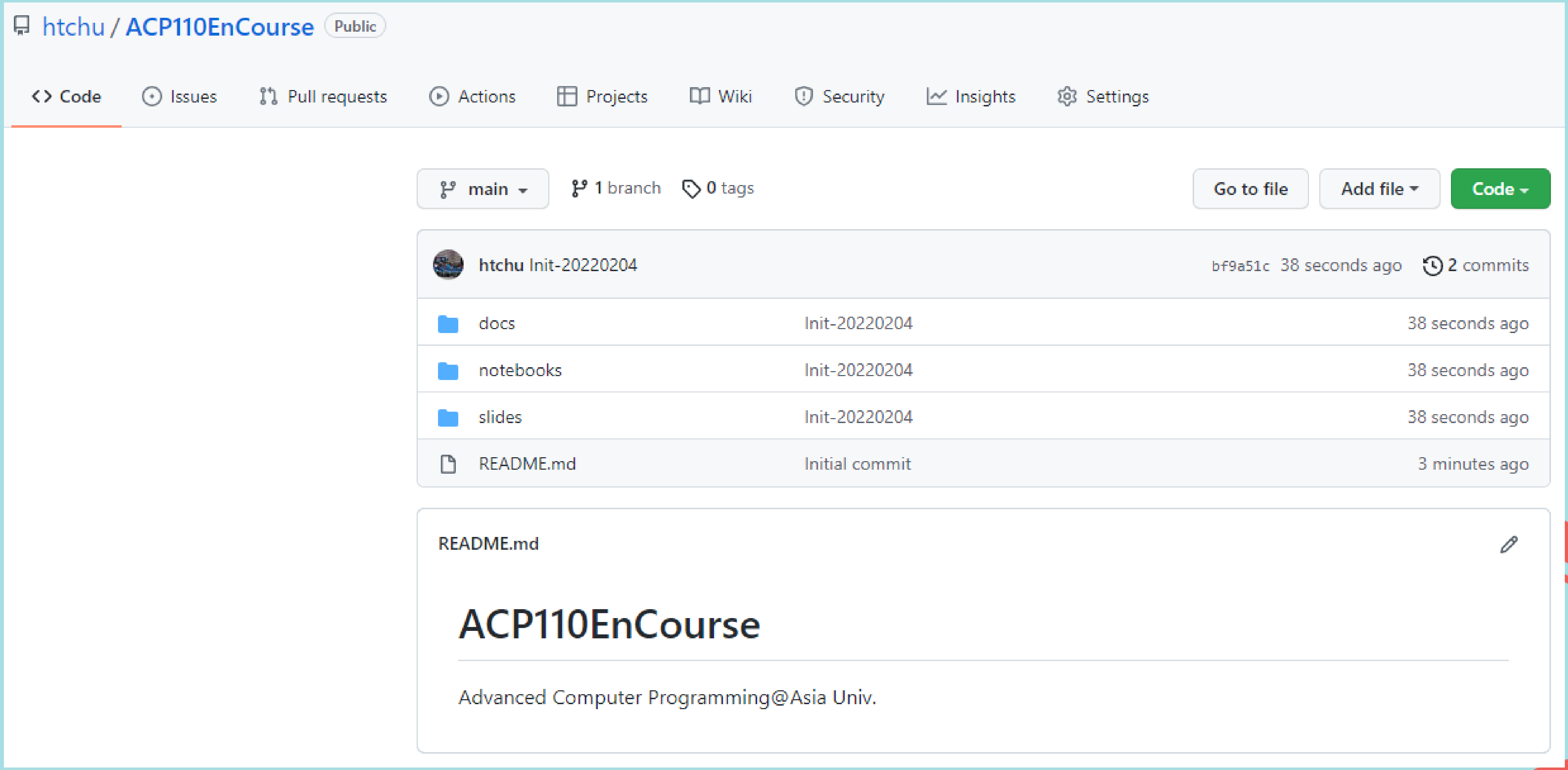
2022-Spring
Advanced Computer Programming (Week 2)

CSIE, Asia Univ.

Course schedule

- W1-Introduction
- W2-Python urllib
- W3-BeautifulSoup
- W4-Web Crawlers
- W5-Scrapy
- W6-Storing Data
- W7-Flask Routes
- W8-Jinja template
- W9-Midterm presentation
- W10-Flask-Mail
- W11-REST API
- W12-AWS Lambda + S3 (1)
- W13-AWS Lambda + S3 (1)
- W14-AWS Glue Data
- W15-AWS Step Functions
- W16-AWS Elastic Beanstalk
- W17-Sample app-Book Recommender
- W18-Final presentation

Course Github


htchu / ACP110EnCourse Public

htchu Init-20220204 bf9a51c 38 seconds ago 2 commits

docs	Init-20220204	38 seconds ago
notebooks	Init-20220204	38 seconds ago
slides	Init-20220204	38 seconds ago
README.md	Initial commit	3 minutes ago

README.md

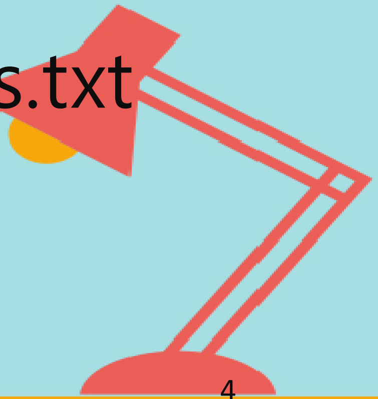
ACP110EnCourse

Advanced Computer Programming@Asia Univ.

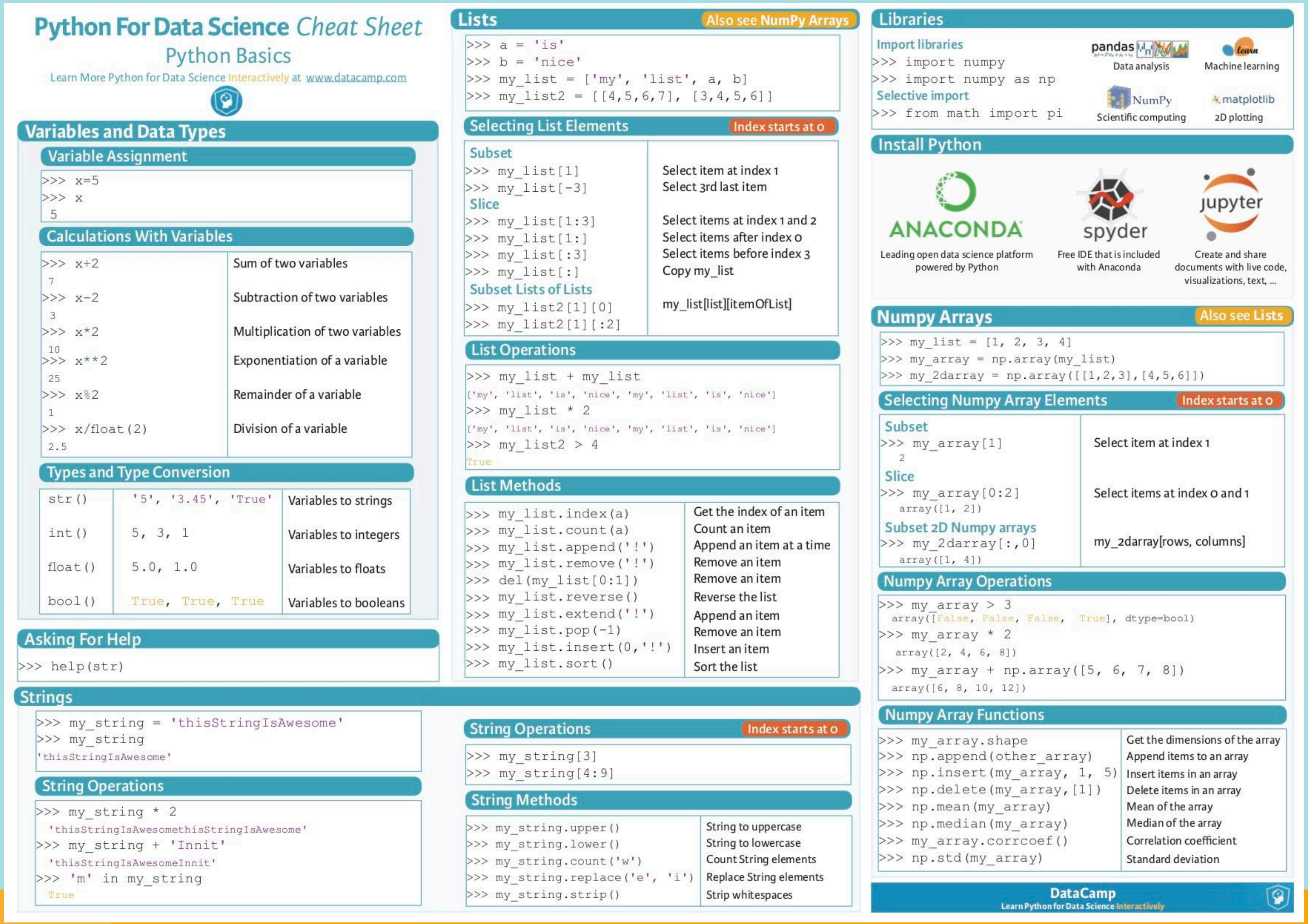
https://github.com/htchu/ACP110EnCourse

Outline

- Python Review
- Building Scrapers
- robots.txt
  - https://developers.google.com/search/docs/advanced/robots/robots_txt
- Assignment 1:
  - scraping the home page of https://focustaiwan.tw/
  - finding and interpreting https://focustaiwan.tw/robots.txt



Python For Data Science Cheat Sheet



Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations — Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Lists — Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements — Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis	learn — Machine learning
NumPy — Scientific computing	matplotlib — 2D plotting

Install Python

ANACONDA	spyder	jupyter
Leading open data science platform powered by Python	Free IDE that is included with Anaconda	Create and share documents with live code, visualizations, text, ...

Numpy Arrays — Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements — Index starts at 0

Subset	
`>>> my_array[1]` → `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

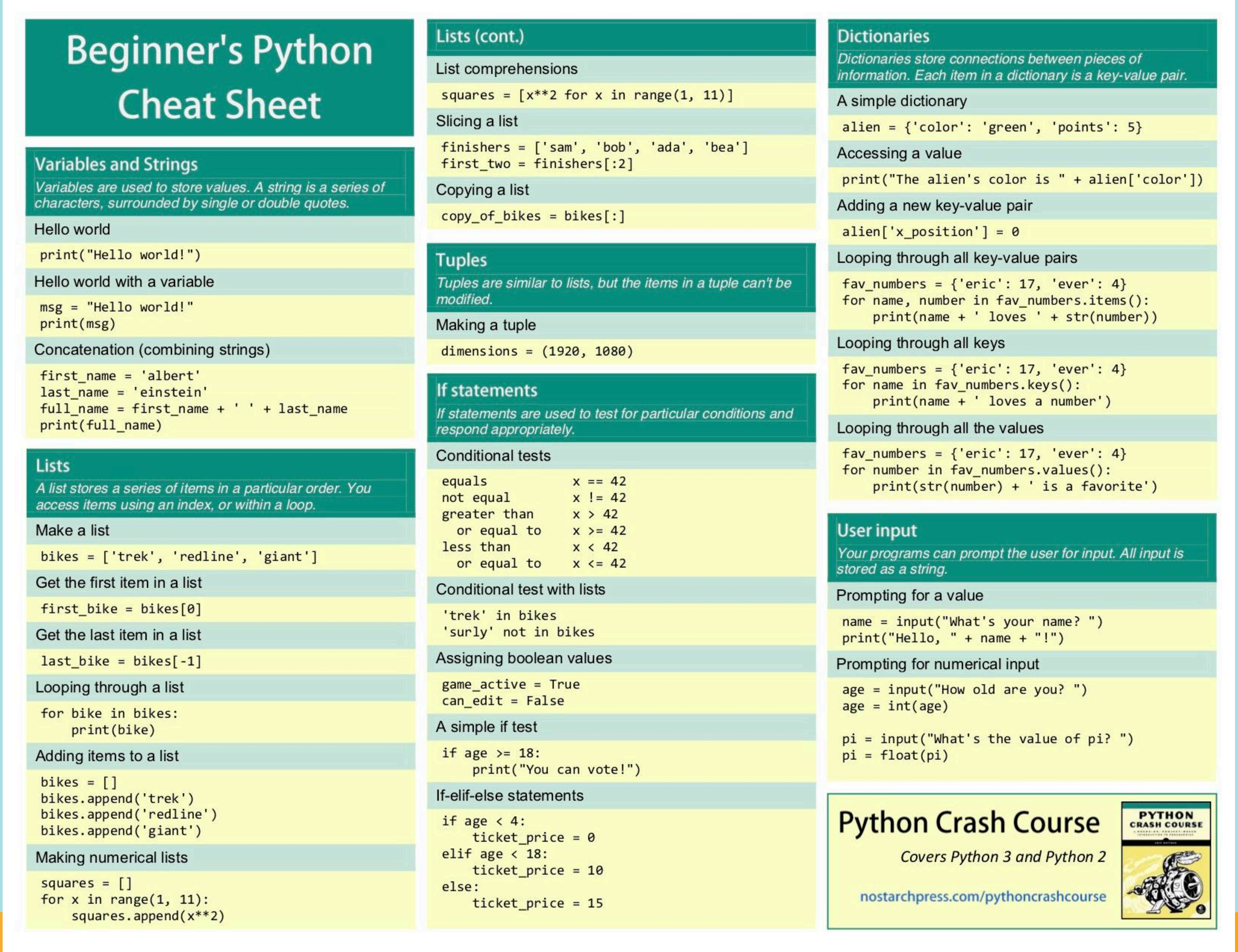
Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False,  True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp
Learn Python for Data Science Interactively



Beginner's Python Cheat Sheet

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

Lists (cont.)

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

Tuples

Tuples are similar to lists, but the items in a tuple can't be modified.

Making a tuple

```
dimensions = (1920, 1080)
```

If statements

If statements are used to test for particular conditions and respond appropriately.

Conditional tests

```
equals                x == 42
not equal             x != 42
greater than          x > 42
   or equal to        x >= 42
less than             x < 42
   or equal to        x <= 42
```

Conditional test with lists

```
'trek' in bikes
'surly' not in bikes
```

Assigning boolean values

```
game_active = True
can_edit = False
```

A simple if test

```
if age >= 18:
    print("You can vote!")
```

If-elif-else statements

```
if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

User input

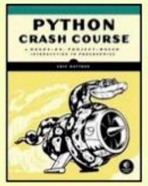
Your programs can prompt the user for input. All input is stored as a string.

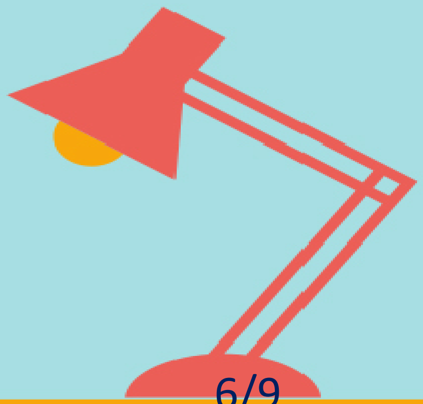
Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

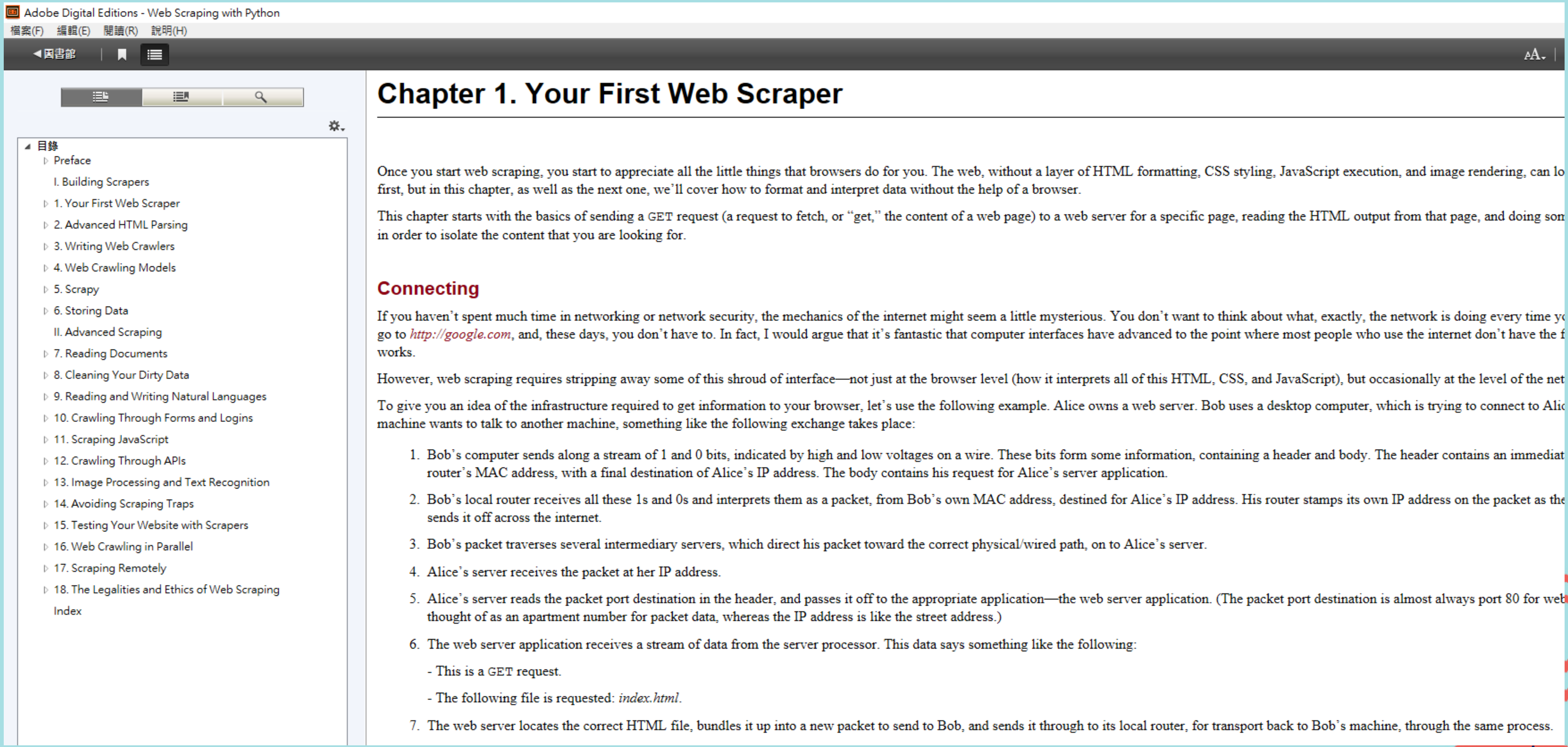
Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)

pi = input("What's the value of pi? ")
pi = float(pi)
```





Building Scrapers


Adobe Digital Editions - Web Scraping with Python
檔案(F) 編輯(E) 閱讀(R) 說明(H)
◀圖書館
AA

Chapter 1. Your First Web Scraper

Once you start web scraping, you start to appreciate all the little things that browsers do for you. The web, without a layer of HTML formatting, CSS styling, JavaScript execution, and image rendering, can lo first, but in this chapter, as well as the next one, we'll cover how to format and interpret data without the help of a browser.

This chapter starts with the basics of sending a `GET` request (a request to fetch, or "get," the content of a web page) to a web server for a specific page, reading the HTML output from that page, and doing son in order to isolate the content that you are looking for.

Connecting

If you haven't spent much time in networking or network security, the mechanics of the internet might seem a little mysterious. You don't want to think about what, exactly, the network is doing every time yo go to *http://google.com*, and, these days, you don't have to. In fact, I would argue that it's fantastic that computer interfaces have advanced to the point where most people who use the internet don't have the f works.

However, web scraping requires stripping away some of this shroud of interface—not just at the browser level (how it interprets all of this HTML, CSS, and JavaScript), but occasionally at the level of the net

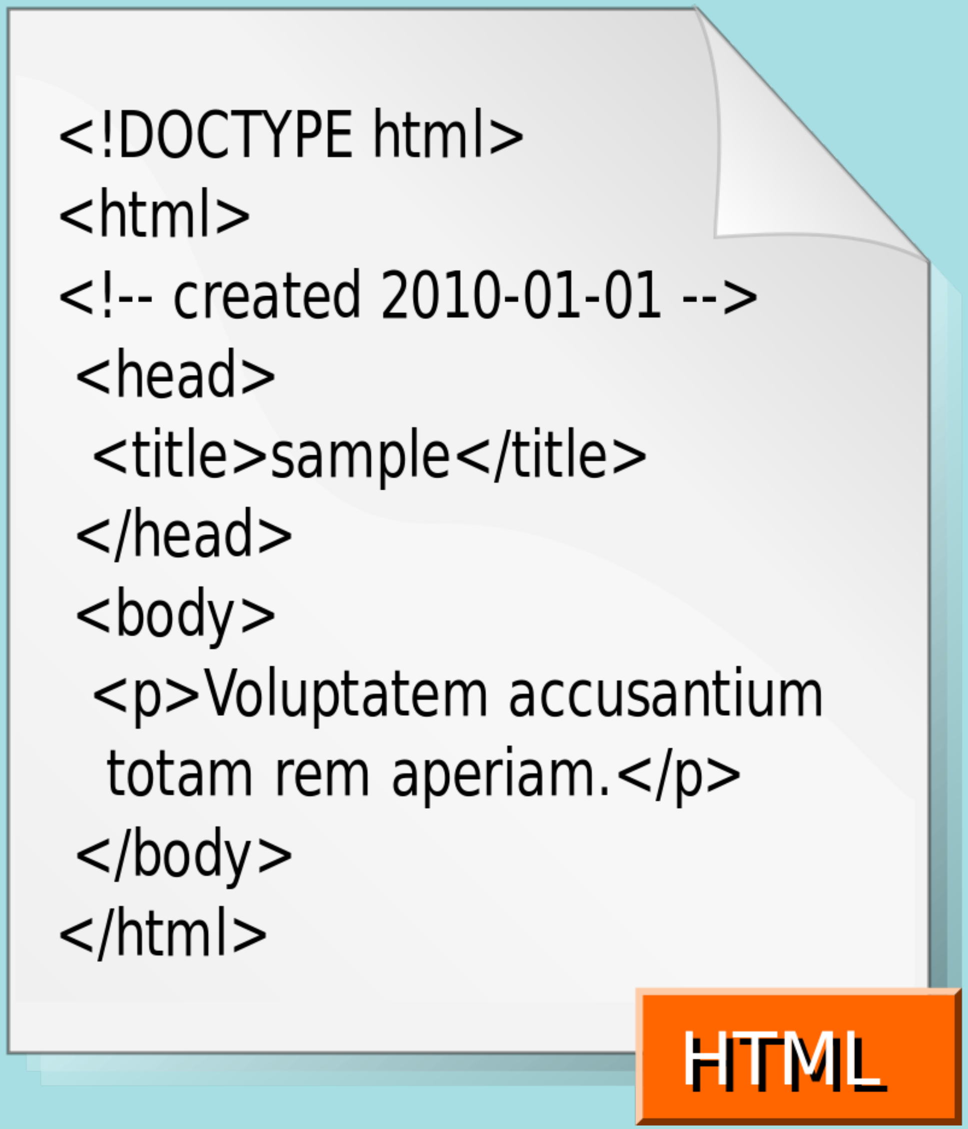
To give you an idea of the infrastructure required to get information to your browser, let's use the following example. Alice owns a web server. Bob uses a desktop computer, which is trying to connect to Alic machine wants to talk to another machine, something like the following exchange takes place:

1. Bob's computer sends along a stream of 1 and 0 bits, indicated by high and low voltages on a wire. These bits form some information, containing a header and body. The header contains an immediat router's MAC address, with a final destination of Alice's IP address. The body contains his request for Alice's server application.
2. Bob's local router receives all these 1s and 0s and interprets them as a packet, from Bob's own MAC address, destined for Alice's IP address. His router stamps its own IP address on the packet as the sends it off across the internet.
3. Bob's packet traverses several intermediary servers, which direct his packet toward the correct physical/wired path, on to Alice's server.
4. Alice's server receives the packet at her IP address.
5. Alice's server reads the packet port destination in the header, and passes it off to the appropriate application—the web server application. (The packet port destination is almost always port 80 for web thought of as an apartment number for packet data, whereas the IP address is like the street address.)
6. The web server application receives a stream of data from the server processor. This data says something like the following:

   - This is a `GET` request.

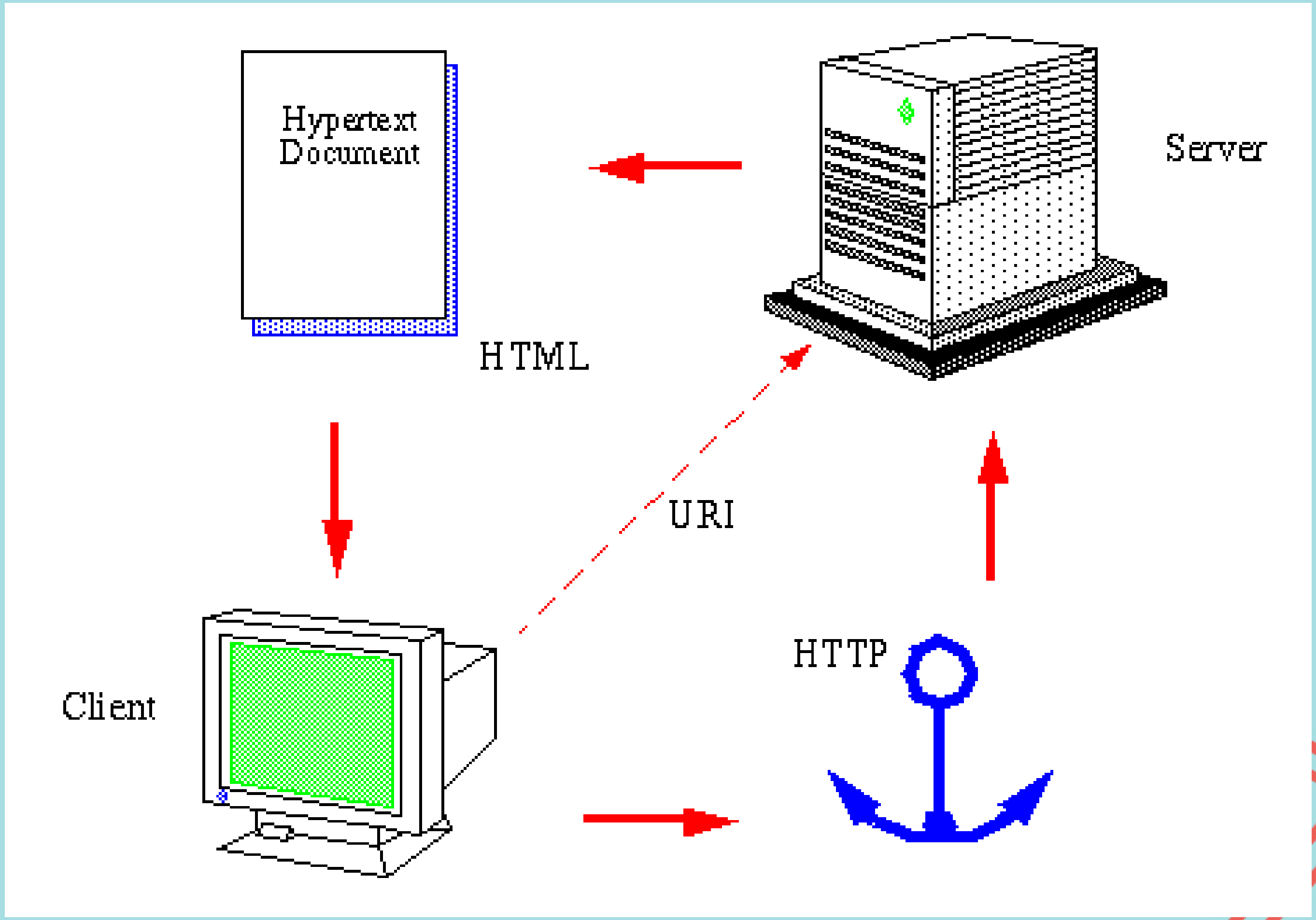
   - The following file is requested: *index.html*.
7. The web server locates the correct HTML file, bundles it up into a new packet to send to Bob, and sends it through to its local router, for transport back to Bob's machine, through the same process.

HTTP, URI, HTML

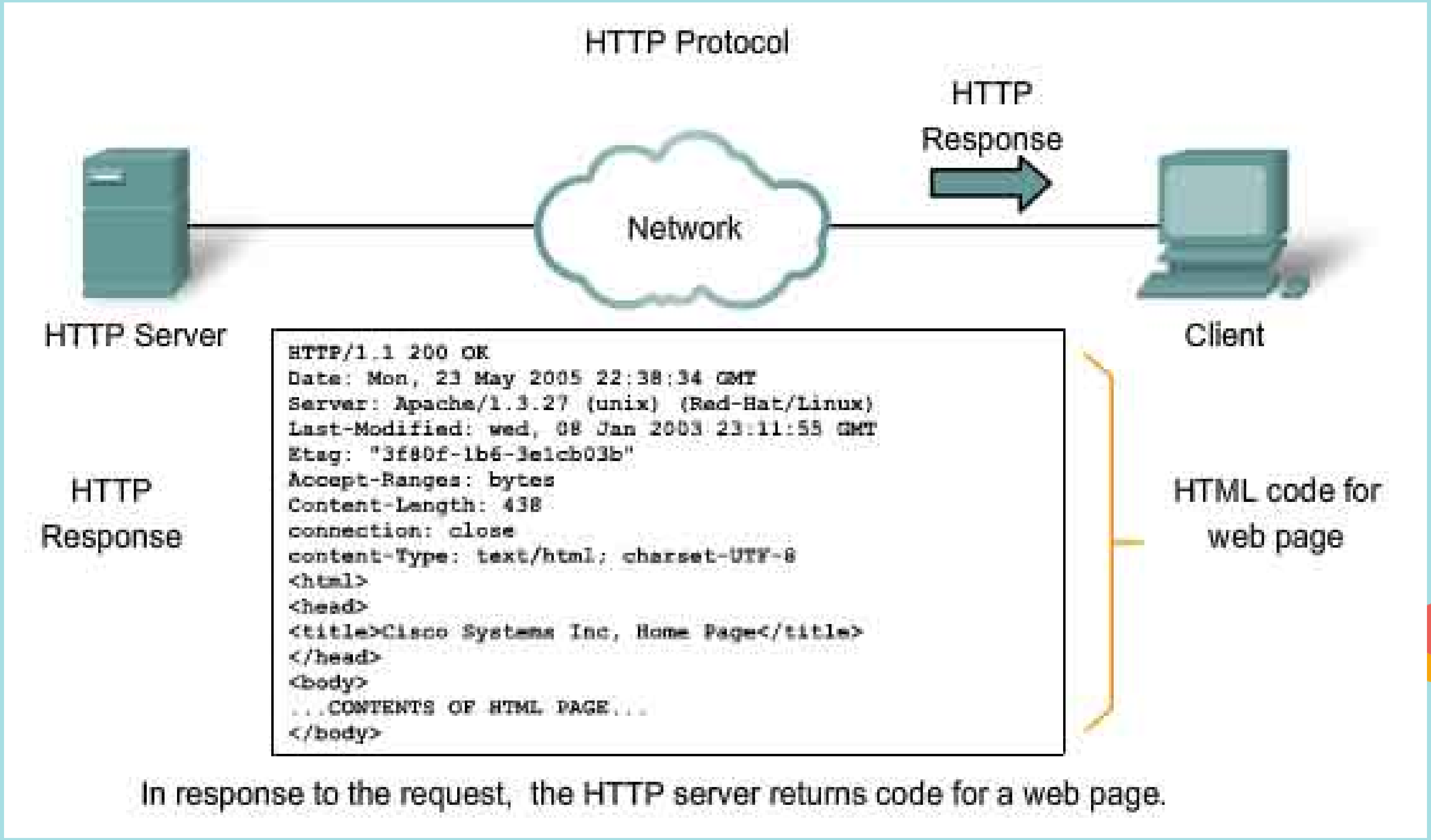


```
<!DOCTYPE html>
<html>
<!-- created 2010-01-01 -->
 <head>
  <title>sample</title>
 </head>
 <body>
  <p>Voluptatem accusantium
   totam rem aperiam.</p>
 </body>
</html>
```

HTML



HTTP



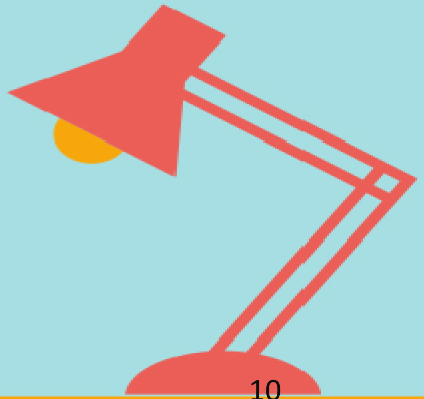
```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (unix) (Red-Hat/Linux)
Last-Modified: wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
connection: close
content-Type: text/html; charset-UTF-8
<html>
<head>
<title>Cisco Systems Inc, Home Page</title>
</head>
<body>
...CONTENTS OF HTML PAGE...
</body>
```

In response to the request, the HTTP server returns code for a web page.

robots.txt

- A robots.txt file is a simple text file containing rules about which crawlers may access which parts of a site. For example, the robots.txt file for example.com may look like this:

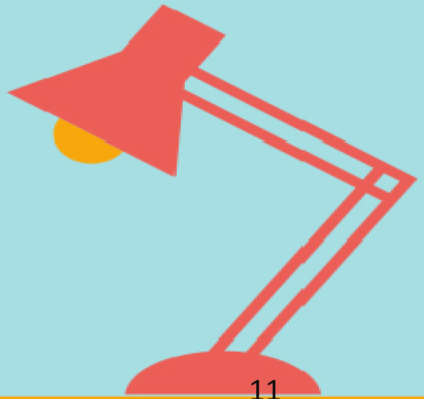
```
# This robots.txt file controls crawling of URLs under https://example.com.
# All crawlers are disallowed to crawl files in the "includes" directory, such
# as .css, .js, but Googlebot needs them for rendering, so Googlebot is allowed
# to crawl them.
User-agent: *
Disallow: /includes/

User-agent: Googlebot
Allow: /includes/

Sitemap: https://example.com/sitemap.xml
```



https://developers.google.com/search/docs/advanced/robots/robots_txt

Assignment 1

- Overview:
  - In this assignment, we will use the basic python web scraping tools urllib and BeautifulSoup to scrape data from the focustaiwan website. Please list the scraped news content and submit it to Tronclass's assignment entry.
- Objectives:
  - Learn how to obtain the content of a web page using web scraping.
  - To explore real html files.
  - Reflect on the possible uses of web scraping capabilities for data science.
- Instructions:
  - Go to the focustaiwan website using any browser. https://focustaiwan.tw/
  - Check the tags in the html content.



Thanks! Q&A