

Unit 1

Programming Fundamentals: Building a Search Engine

BSCS/BSDS/MCS-EA 1st

Week2—02-March-2020

Muhammad Ateeq, Asst. Prof. of CS @ The IUB

WHAT:: In this unit we will be writing a program to extract the first link from a given web page.

Introducing the Web Crawler

A **web crawler** is a program that collects content from the web. A web crawler finds web pages by starting from a seed page and following links to find other pages, and following links from the other pages it finds, and continuing to follow links until it has found many web pages. Here is the process that a web crawler follows:

Start from one preselected page. We call the starting page the "seed" page.

Extract all the links on that page. (This is the part we will work on in this unit and Unit 2.)

Follow each of those links to find new pages.

Extract all the links from all of the new pages found.

Follow each of those links to find new pages.

Extract all the links from all of the new pages found

...

This keeps going as long as there are new pages to find, or until it is stopped. In this unit we will be writing a program to extract the first link from a given web page. In Unit 2, we will figure out how to extract all the links on a web page. In Unit 3, we will figure out how to keep the crawl going over many pages.

Quiz

What is the goal of Unit 1?

- Get started programming.
- Learn important computers science concepts.
- Write code to extract a link from a web page.
- Write code to rank web pages.

Programming

A **computer** is a machine that can execute a program. With the right program, a computer can do any mechanical computation you can imagine.

A **program** describes a very precise sequence of steps. Since the computer is just a machine, the program must give the steps in a way that can be executed mechanically. That is, the program can be followed without any thought.

A **programming language** is a language designed for producing computer programs. A good programming language makes it easy for humans to read and write programs that can be executed by a computer.

Python is a programming language. The programs that we write in the Python language will be the input to the Python interpreter, which is a program that runs on the computer. The Python interpreter reads our programs and executes them by following the rules of the Python language.

Quiz

What is a programming language?

- a. a language designed to be executed by computers
- b. a language designed for describing programs
- c. a language designed to be written by humans, and executed by computers
- d. a language designed to be read by humans, and written by computers
- e. a language designed to be read and written by humans, and executed by computers

Getting Started with Python Programming

We have already installed IDLE (A Python GUI) for our programming experience. Let's get started:

- `print(3)`
- `1+1`
- `3*4`
- `10/2`
- `52 * 3 + 12 * 9`
- `(52 * 3) + (12 * 9)`
- `52 * (3 + 12) * 9`

For example, this code prints out the number of seconds in a year:

- `print(365 * 24 * 60 * 60)`

Quiz

Write a Python program that prints out the number of minutes in seven weeks.

Unit 1-02

Week2—03-March-2020

Programming Language

Why do we need to invent and learn new languages, like Python, to program computers, rather than using natural languages like English or Mandarin?

There are many reasons why a designed language like Python is better for writing programs than a natural language like English. One problem with natural languages is that they are ambiguous. Hence, not everyone will interpret the same phrase the same way. To program computers, it is important that we know exactly what our programs mean, and that the computer will run them with the meaning we intended. Another problem with natural language is that they are very verbose. To say something with the level of precision needed for a computer to be able to follow it mechanically would require an awful lot of writing. We want our programs to be short so it is less work to write them, and so that it is easier to read and understand them.

Grammar

Compared to a natural language, like English, programming languages like Python adhere to a strict grammatical structure. In English, even if a phrase is written or spoken incorrectly, it can still be understood with the help of context or other cues. On the other hand, in a programming language like Python, the code must match the language grammar exactly. The Python interpreter has no idea what to do with input that is not in the Python language, so it produces an error.

Basic English Grammar Rules:

Sentence → *Subject Verb Object*

Subject → *Noun*

Object → *Noun*

Verb → **Eat**

Verb → **Like**

Noun → **I**

Noun → **Python**

Noun → **Cookies**

When programming language grammar is not followed the interpreter will return a "**SyntaxError**" message. This means that the structure of the code is inconsistent with the rules of the programming language.

Quiz

Which of these sentences can be produced from this grammar, starting from *sentence*?

- a. **Python Eat Cookies**
- b. **Python Eat Python**
- c. **I Like Eat**

Python Grammar for Arithmetic Expressions

An **expression** is something that has a value. Here are some examples of expressions in Python:

```
3
1 + 1
7 * 7 * 24 * 60
```

Here is one of the rules of the Python grammar for making expressions:

Expression → *Expression Operator Expression*

This is an example of a **recursive definition**.

Here are some of the Python grammar rules for arithmetic expressions:

Expression → *Expression Operator Expression*

Expression → *Number*

Operator → **+**

Operator → *****

Number → **0, 1, ...**

We need to add one more rule to our expression grammar to be able to produce all of the expressions we have used so far:

Expression → **(Expression)**

Quiz

Which of the following are valid Python expressions that can be produced starting from *Expression*?

There may be more than one.

- a. 3
- b. ((3))
- c. (1 * (2 * (3 * 4)))
- d. + 3 3
- e. (((7)))

Quiz

Write Python code to print out how far light travels in centimeters after one nanosecond using the multiplication operator.

The speed of light is 299792458 meters per second.

One meter is 100 centimeters.

One nanosecond is one billionth (1/1000000000) of a second.

Variables

A **variable** is a name refers to a value. In Python, we can use any sequence of letters and numbers and underscores (_) we want to make a variable name, so long as it does not start with a number. Here are some examples of valid variable names:

```
processor_speed
n
hours
item73
```

To introduce a new variable, we use an **assignment statement**:

Credits: Dave Evans and Udacity

Name = Expression

After executing an assignment expression, the name refers to the value of the expression on the right side of the assignment:

```
speed_of_light = 299792458
```

We can use the variable name anywhere we want and it means the same things as the value it refers to. Here is an expression using the name to print out the speed of light in centimeters:

```
print(speed_of_light * 100)
```

Quiz

Write Python code to print out how far light travels in a light year.

```
print(speed_of_light * 365 * 24 * 60 * 60)
```

We can make a variable that contains the numbers of seconds in a year: `secs_in_year`

```
secs_in_year = 365 * 24 * 60 * 60  
Print(speed_of_light * secs_in_year)
```

Variables Can Vary

The value a variable refers to can change. When a variable name is used, it always refers to the last value assigned to that variable.

```
shoe_price = 3000 #before sales period starts  
shoe_price = 2000 #after sales period starts
```

This gets more interesting when we use the same variable on both sides of an assignment. The right side is evaluated first, using the current value of the variable. Then the assignment is done using that value. In the following expressions, the value of days changes from 49 to 48 and then to 47 as the expression changes:

```
days = 7 * 7 # after the assignment, days refers to 49  
days = 48 # after the assignment, days refers to 48  
days = days - 1 # after the assignment, days refers to 47  
days = days - 1 # after the assignment, days refers to 46
```

It is important to remember that although we use = for assignment it does not mean equality. You should think of the = sign in Python as an arrow, ←, showing that the value the right side evaluates to is being assigned to the variable name on the left side.

Quiz

What is the value of hours after running this code:

```
hours = 9  
hours = hours + 1  
hours = hours * 2
```

a. 9 b. 10 c. 18 d. 20 e. 22 f. Error

Unit 1-03

Week3—09-March-2020

Defining Variables

For Python to be able to output a result, we need to always define a variable by assigning a value to it before using it.

```
minutes = 30
minutes = minutes + 1
seconds = minutes * 60
print(seconds)
1860
```

Quiz

Write Python code that defines the variable age to be your age in years, and then prints out the number of days you have been alive. If you don't want to use your real age, feel free to use your age in spirit instead.

```
age = 26
days_per_year = 365
days_alive = age * days_per_year
print(days_alive)
9490
```

Strings

A **string** is a sequence of characters surrounded by quotes; either single or double.

```
'I am a string!'
```

The only requirement is that the string must start and end with the same kind of quote.

```
"I prefer double quotes!"
```

This allows you to include quotes inside of quotes as a character in the string.

```
"I'm happy I started with a double quote!"
```

Using the interpreter, notice how the color of the input changes before and after you put quotes on both sides of the string. What happens when you do not include any quotes:

```
print(Hello)
```

Without the quotes, Python reads Hello as a variable that is undefined:

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

As we saw above, Python will not print an undefined variable, which is why we get the name error.

Quiz

Which of the following are valid strings in Python?

- a. "Ada"
- b. 'Ada"
- c. "Ada
- d. Ada
- e. "'Ada'

Quiz

Define a variable, **name**, and assign to it a string that is your name.

Using Operators on Strings

We can also use the plus operator (+) on strings, but it has a different meaning from when it is used on numbers. With string, plus means **concatenation**.

<string> + <string> outputs the concatenation of the two strings

Try concatenating the string **'Hello'** to **name**.

You can create a space between the strings by adding a space to one of the strings. You can also continue to add strings as many times as you need.

```
name = 'Dave'
print('Hello ' + name + '!' + '!' + '!')
Hello Dave!!!
```

However, you cannot use the plus operator to combine strings and integers, as in the case of:

```
print('My name is ' + 9)
```

When you run this program you should see an error message like this:

TypeError: cannot concatenate 'str' and 'int' objects.

It is a bit surprising that you can multiply strings and integers!

```
print('!' * 12)
!!!!!!!!!!!!
```

This program multiplies the string by the integer to return 12 exclamation points!

Indexing Strings

When you want to select sub-sequences from a string, it is called **indexing**. Use the square

brackets `[]` to specify which part of the string you want to select.

`<string>[<expression>]`

For example, if we have the string `'bcsbmorning'` and we want to select the character in the zero position (that is, the first character), we would write:

```
'bcsbmorning'[0]
```

The positions in a string are numbered starting with 0, so this evaluates to `'b'`.

Indexing strings is the most useful when the string is given using a variable:

```
'bcsbmorning' [0] → 'b'
'bcsbmorning' [1 +1 ] → 'c'
name = 'Dave'
name[0] → 'D'
```

When you use negative numbers in the index it starts counting from the back of the string:

```
name = 'Dave'
print(name[-1])
e
```

or,

```
name='Dave'
print(name[-2])
v
```

When you try to index a character in a position where there is none, Python produces an error indicating that the index is out of range:

```
name = 'Dave'
print(name[4])
IndexError: string index out of range
```

Quiz

Given the variable,

```
s = '<any string>'
```

which of these pairs are two things with the exact same value?

- a. `s[3]`, `s[1+1+1]`
- b. `s[0]`, `(s+s)[0]`
- c. `s[0] + s[1]`, `s[0+1]`
- d. `s[1]`, `(s + 'ity')[1]`
- e. `s[-1]`, `(s + s)[-1]`

Selecting Sub-Sequences from String

You can select a sub-sequence of a string by designating a starting position and an end position. Python reads the characters positions starting at 0, so that if we consider the string 'udacity' that has 7 characters, there are 6 positions with 'u' being in the 0 position.

`<string>[<expression>]` → a one-character string
`<string>[<start expression>:<stop expression>]` → a string that is a sub-sequence of the characters in the string, from the start position up to the character before the stop position. If the start expression is missing, the sub-sequence starts from the beginning of the string; if the stop expression is missing, the sub-sequence goes to the end of the string.

Examples:

```
word = 'assume'
print(word[3])
u
print(word[4:6])
me
print(word[4:])
me
print(word[:2])
as
print(word[:])
assume
```

Quiz

Write Python code that prints out Udacity (with a capital U), given the definition

```
s = 'audacity'
```

Quiz

For any string,

```
s = '<any string>'
which of these is always equivalent to s:
a. s[:]
b. s + s[0:-1 + 1]
c. s[0:]
d. s[:-1]
e. s[:3] + s[3:]
```

Finding Strings in Strings

The **find method** is a built in operation, or method, provided by Python, that operates on strings. The output of find is the position of the string where the specified sub-string is found.

<search string>.find(<target string>)

If the *target string* is not found anywhere in the *search string*, then the output will be -1. Here are some examples (try them yourself in the interpreter):

```
pythagoras = 'There is geometry in the humming of the strings, there is
music in the spacing of the spheres. '
print(pythagoras.find('string'))
40
print(pythagoras[40:])
strings, there is music in the spacing of the spheres.
print(pythagoras.find('T'))
0
print(pythagoras.find('sphere'))
86
print(pythagoras[86:])
spheres
print(pythagoras.find('algebra'))
-1
```

Quiz

Which of the following evaluate to -1:

- a. 'test'.find('t')
 - b. "test".find('st')
 - c. "Test".find('te')
 - d. 'west'.find('test')
- 17

Quiz

For any string,

s = '<any string>'

which of the following always has the value 0?

- a. s.find(s)
- b. s.find('s')
- c. 's'.find('s')
- d. s.find('')
- e. s.find(s + '!!!') + 1

Using find with Numbers

In addition to passing in a target string to find, we can also pass in a number:

<search string>.find(<target string>, <number>)

The *number* input is the position in the *search string* where find will start looking for the *target string*. So, the output is a number giving the position of the first occurrence of the target string in the search string at or after the number input position. If there is no occurrence at or after that position, the output is -1. For example:

```
motto = "We can do programming, yes I can, yes you can."
print(motto.find('can'))
3
```

```

print(motto.find('can', 0))
3
print(motto.find('can', 3))
3
print(motto.find('can', 4))
29
print(motto.find('can', 29))
29
print(motto.find('can', 30))
42
print(motto.find('can', 43))
-1

```

Quiz

For any variables *s* and *t* that are strings, and *i* that is a number:

s = '<any string>'

t = '<any string>'

i = <any number>

which of these are equivalent to *s.find(t,i)*:

a. *s[i:].find(t)*

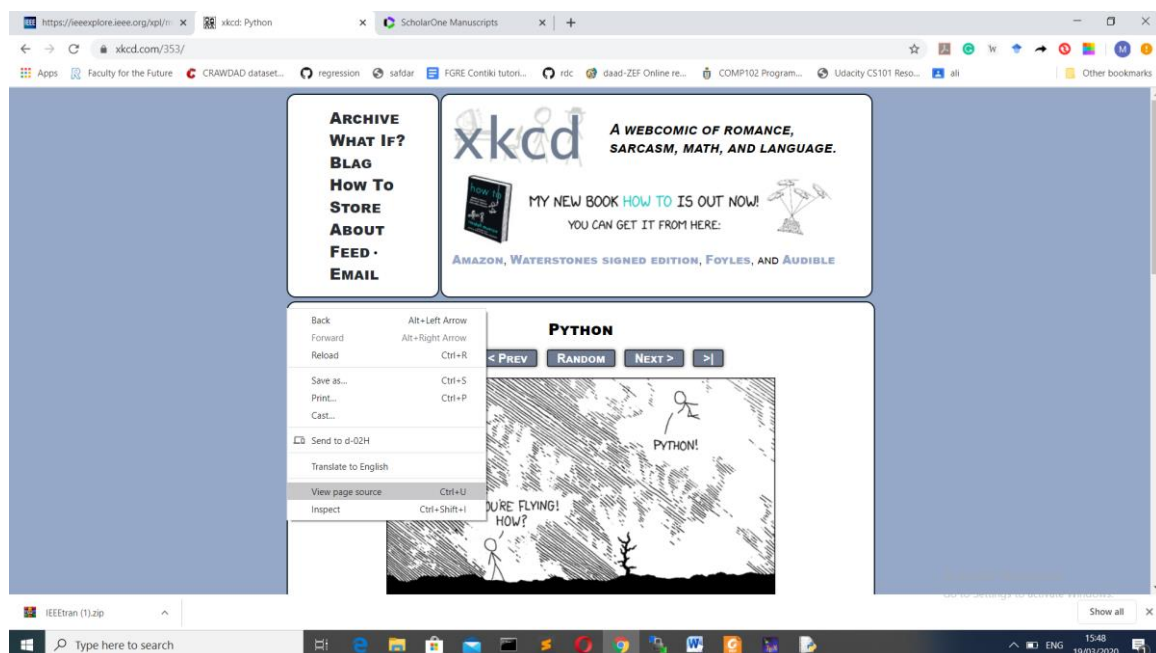
b. *s.find(t)[:i]*

c. *s[i:].find(t) + i*

d. *s[i:].find(t[i:])*

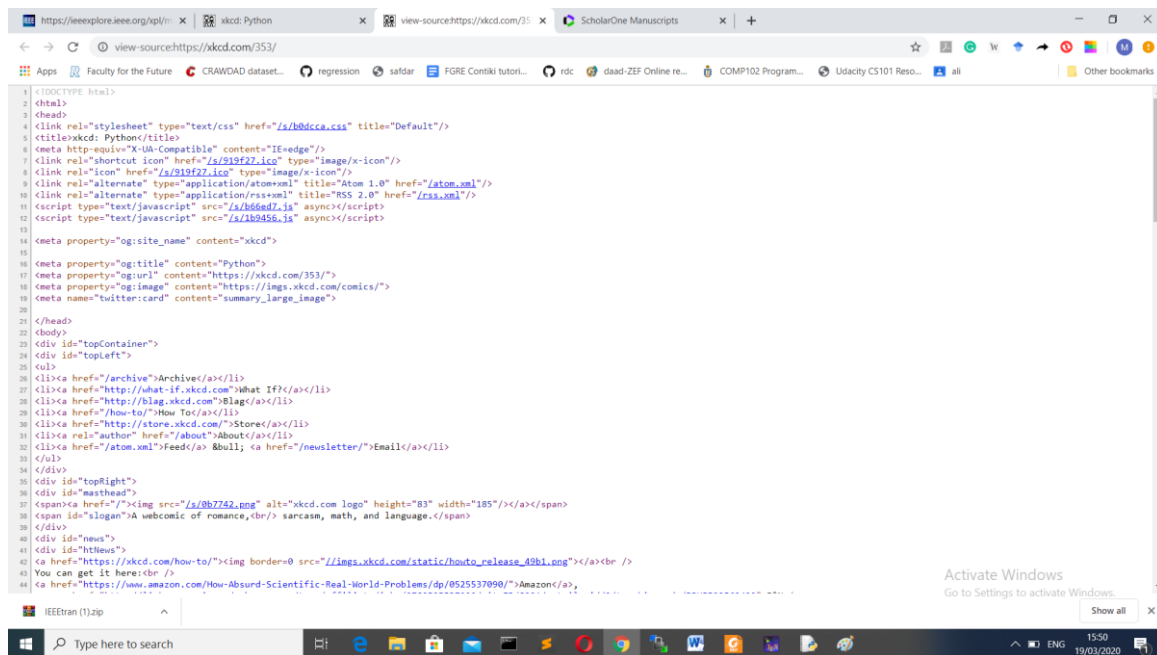
Extracting Links

A **web page** is really just a long string of characters. Your **browser** renders the web page in a way that looks more attractive than just the string of characters. You can view the string of characters for any web page in your browser. How to do this depends on the browser you are using. For Chrome and Firefox, right-click anywhere on the page that is not a link and select "View Page Source". For Internet Explorer, right-click and select "View Source". Here's what this looks like in Chrome:



Credits: Dave Evans and Udacity

Select **View Page Source** from the menu. The raw string for the web page pops up in a new window:



For our web crawler, the important thing is to find the links to other web pages in the page. We can find those links by looking for the anchor tags that match this structure:

```
<a href="url">
```

For example, here is a link to the **News/Blag** page:

```
<a href=http://blog.xkcd.com/>
```

To build our crawler, for each web page we want to find all the link target URLs on the page. We want to keep track of them and follow them to find more content on the web. For this unit, we will do the first step which is to extract the first target URL from the page. In Unit 2, we will see how to keep going to get all the link targets, and in Unit 3, we will see how to keep track of them to be able to crawl the target pages. For now, our goal is to take the text from a web request and find the first link target in that text. We can do this by finding the anchor tag, `<a href=`, and then extract from that tag the URL that is found between the double quotes. We will assume that with the page's contents in a variable, **page**.

Quiz

Write Python code that initializes the variable **start_link** to be the value of the position where the first `<a href=` occurs in the string **page** refers to.

```
page = <contents of a web page>
start_link = page.find('<a href= ')
```

Quiz

Write Python code that assigns to the variable **url** a string that is the value of the first URL that appears in a link tag in the string **page**. For the example page used in the test code, your code should end up with **url** having the value `'http://udacity.com'`.

```
page = <contents of a web page>
start_link = page.find('<a href= ')
start_quote = page.find('"',start_link)
end_quote = page.find('"',start_quote+1)
```

Credits: Dave Evans and Udacity

```
url = page[start_quote+1:end_quote]
print(url)
http://udacity.com
```

Note: If you have not been able to initialize the **page** variable, use some content from the source of xkcd.com/353 and enclose it in triple quotes (a triple quote can be placed using the q= single quote typed three times without a space). For example try this:

```
page = '''
<body>

<div id="topContainer">
<div id="topLeft">
<ul>
<li><a href="/archive">Archive</a></li>
<li><a href="http://what-if.xkcd.com">What If?</a></li>
<li><a href="http://blag.xkcd.com">Blag</a></li>
<li><a href="/how-to/">How To</a></li>
<li><a href="http://store.xkcd.com/">Store</a></li>
<li><a rel="author" href="/about">About</a></li>
<li><a href="/atom.xml">Feed</a> &bull; <a
href="/newsletter/">Email</a></li>
</ul>
</div>
'''
```

Yay! You did it and are off to a great start!

You've learned about programs, variables, expressions, and strings, and a well on your way to building a web crawler. Next unit, we will learn some big ideas in computer science that will make this code more useful and enable us to get all the links on the page, not just the first one. ☺