

Unit 1: How to get started: your first program

Contents

- 1 [Unit 1: How to get started: your first program](#)
 - 1.1 [Introducing the Web Crawler](#)
 - 1.1.1 [Quiz 1: First Quiz](#)
 - 1.2 [Programming](#)
 - 1.2.1 [Quiz 2: What is a Programming Language](#)
 - 1.3 [Getting Started with Python Programming](#)
 - 1.3.1 [Quiz 3: First Programming Quiz](#)
 - 1.4 [Would You Rather](#)
 - 1.5 [Grammar](#)
 - 1.6 [Backus-Naur Form](#)
 - 1.6.1 [Quiz 4: Eat Quiz](#)
 - 1.7 [Python Expressions](#)
 - 1.7.1 [Quiz 5: Python Expressions](#)
 - 1.7.2 [Quiz 6: Speed of Light](#)
 - 1.8 [Admiral Grace Hopper \(1906- 1992\)](#)
 - 1.9 [Variables](#)
 - 1.9.1 [Quiz 7: Variables](#)
 - 1.10 [Variables Can Vary](#)
 - 1.10.1 [Quiz 8: Varying Variables Quiz 1](#)
 - 1.10.2 [Quiz 9: Varying Variables Quiz 2](#)
 - 1.10.3 [Quiz 10: Spirit Age](#)
 - 1.11 [Strings](#)
 - 1.11.1 [Quiz 11: Valid Strings](#)
 - 1.12 [Ada](#)
 - 1.12.1 [Quiz 12: Hello!!!](#)
 - 1.12.2 [Strings and Numbers](#)
 - 1.13 [Indexing Strings](#)
 - 1.13.1 [Quiz 13: Same Value](#)
 - 1.14 [Selecting Sub-Sequences](#)
 - 1.14.1 [Quiz 14: Capital Udacity](#)
 - 1.14.2 [Quiz 15: Understanding Selection](#)
 - 1.15 [Finding Strings in Strings](#)
 - 1.15.1 [Quiz 16: Testing](#)
 - 1.15.2 [Quiz 17: Testing 2](#)
 - 1.16 [Find with Numbers](#)
 - 1.16.1 [Quiz 18: Find with Numbers](#)
 - 1.17 [Extracting Links](#)
 - 1.17.1 [Quiz 19: Extracting Links](#)
 - 1.17.2 [Quiz 20: Final Quiz](#)

- 1.18 [Answer Key](#)

- 1.18.1 [Quiz 1: Answer](#)
- 1.18.2 [Quiz 2: Answer](#)
- 1.18.3 [Quiz 3: Answer](#)
- 1.18.4 [Quiz 4: Answer](#)
- 1.18.5 [Quiz 5: Answer](#)
- 1.18.6 [Quiz 6: Answer](#)
- 1.18.7 [Quiz 7: Answer](#)
- 1.18.8 [Quiz 8: Answer](#)
- 1.18.9 [Quiz 9: Answer](#)
- 1.18.10 [Quiz 10: Answer](#)
- 1.18.11 [Quiz 11: Answer](#)
- 1.18.12 [Quiz 12: Answer](#)
- 1.18.13 [Quiz 13: Answer](#)
- 1.18.14 [Quiz 14: Answer](#)
- 1.18.15 [Quiz 15: Answer](#)
- 1.18.16 [Quiz 16: Answer](#)
- 1.18.17 [Quiz 17: Answer](#)
- 1.18.18 [Quiz 18: Answer](#)
- 1.18.19 [Quiz 19: Answer](#)
- 1.18.20 [Quiz 20: Answer](#)

*[

!***[strong text]*****[1]**

][1]*

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 you will be writing a program to extract the first link from a given web page. In Unit 2, you will find out

how to extract all the links on a web page. In Unit 3, you will see how to keep the crawl going over many pages.

Quiz 1: First Quiz

What is the goal of Unit 1 ?

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

[Answer](#)

Programming

Programming is the core of computer science.

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 2: What is a Programming Language

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

[Answer](#)

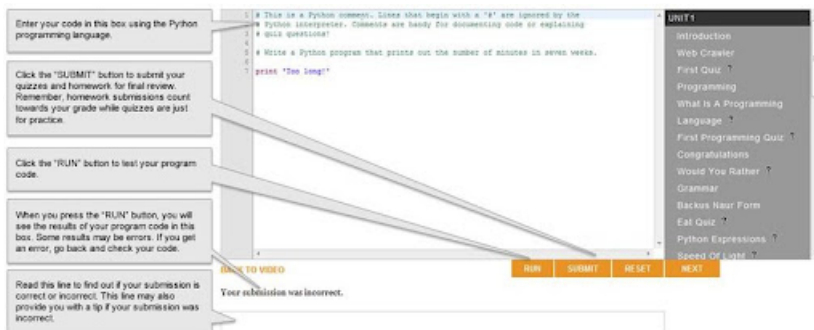
Getting Started with Python Programming

We have provided a way for you to run Python programs using your web browser. You do not need to install any extra programs to do this.

There are two parts to the Python programming environment you will see in your web browser:

1. The top part is an editor, where you can write and edit code.
2. The bottom part is an output window, where you can see the results of running your code.

To try running your code, click the "Run" button below the editor.



Find the value of something in Python by using print like this:

```
print 3
3
```

Use a Python expression to return a value. Here are some examples:

```
1 + 1    #addition
2 - 1    #subtraction
2 * 6    #multiplication
```

You can compose expressions to make more complicated expressions, such as:

```
52 * 3 + 12 * 9
```

You can also use parentheses to group expressions:

```
(52 * 3) + (12 * 9)
```

which is different from:

```
52 * (3 + 12) * 9
```

The code below prints out the number of seconds in a year:

```
print 365 * 24 * 60 * 60
31536000
```

Quiz 3: First Programming Quiz

Write a Python program that prints out the number of minutes in seven weeks. You should enter your program in the editor window (top part), and then click "'Run'" to see the output in the bottom part.

Answer

Would You Rather

Why should you 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. You want your 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, programming languages 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 $\hat{+}$ Subject Verb Object
- Subject $\hat{+}$ Noun
- Object $\hat{+}$ Noun
- Verb $\hat{+}$ **Eat**
- Verb $\hat{+}$ **Like**
- Noun $\hat{+}$ **I**
- Noun $\hat{+}$ **Python**
- Noun $\hat{+}$ **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.

Backus-Naur Form

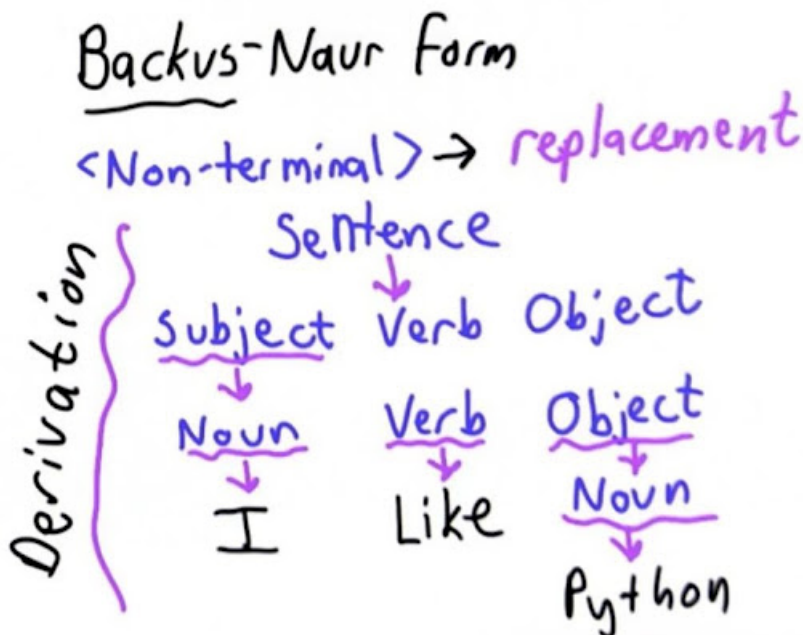
The notation we used to describe the grammar is known as **Backus-Naur Form**, which was introduced in the 1950s by John Backus, the lead designer of the Fortran programming language at IBM.

The purpose of Backus-Naur Form is to describe a programming language in a simple and concise manner. The structure of this form is:

<Non-Terminal> \rightarrow replacement

The replacement can be any sequence of zero or more non-terminals or terminals.

Terminals never appear on the left side of a rule. Once you get to a terminal there is nothing else you can replace it with. Here is an example showing how to derive a sentence by following the replacement rules:



- **Sentence** \rightarrow Subject Verb Object
 - \rightarrow Noun Verb Object
 - \rightarrow I Verb Object
 - \rightarrow I Like Object
 - \rightarrow I Like Noun
 - \rightarrow I Like Python

The important thing about a replacement grammar is that we can describe an infinitely large language with a small set of precise rules.

Quiz 4: Eat 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

Answer

Python 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 $\hat{+}$ *Expression* *Operator* *Expression*

The *Expression* non-terminal that appears on the left side can be replaced by an *Expression*, followed by an *Operator*, followed by another *Expression*. For example, $1 + 1$ is an *Expression Operator Expression*.

The interesting thing about this rule is that it has *Expression* on both the left and right sides! This looks circular, and would be, except we also have other rules for *Expression* that do not include *Expression* on the right side. This is an example of a **recursive definition**. To make a good recursive definition you need at least two rules:

1. A rule that defines something in terms of itself.

- *Expression* $\hat{+}$ *Expression* *Operator* *Expression*

2. A rule that defines that thing in terms of something else that we already know.

- *Expression* $\hat{+}$ *Number*

Recursive definitions are a very powerful idea in computer science. They allow us to define infinitely many things using a few simple rules. You will see this more in Unit 6.

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

- *Expression* $\hat{+}$ *Expression* *Operator* *Expression*
- *Expression* $\hat{+}$ *Number*
- *Operator* $\hat{+}$ $+$
- *Operator* $\hat{+}$ $*$
- *Number* $\hat{+}$ $0, 1, \dots$

Here is an example derivation using this grammar:

- *Expression* $\hat{+}$ *Expression* *Operator* *Expression*
- $\hat{+}$ *Expression* $+$ *Expression*
- $\hat{+}$ *Expression* $+$ *Number*
- $\hat{+}$ *Expression* $+$ 1
- $\hat{+}$ *Expression* *Operator* *Expression* $+$ 1
- $\hat{+}$ *Number* *Operator* *Expression* $+$ 1
- $\hat{+}$ 2 *Operator* *Expression* $+$ 1
- $\hat{+}$ $2 * \text{Expression}$ $+$ 1
- $\hat{+}$ $2 * \text{Expression}$ *Operator* *Expression* $+$ 1
- $\hat{+}$ $2 * \text{Number}$ *Operator* *Expression* $+$ 1
- $\hat{+}$ $2 * 3$ *Operator* *Expression* $+$ 1

- $2 * 3 * \text{Expression} + 1$
- $2 * 3 * \text{Number} + 1$
- $2 * 3 * 3 + 1$
- (Note: the example here is slightly different than the one in the video. The $3+3$ expression has been changed to $3*3$, since the precedence rules in Python would have grouped $2 * 3 + 3 + 1$ as $(2 * 3) + 3 + 1$, so it would not be interpreted as shown in the derivation.)

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:

- $\text{Expression} \rightarrow (\text{Expression})$

Quiz 5: Python Expressions

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)))

Answer

Quiz 6: Speed of Light

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.

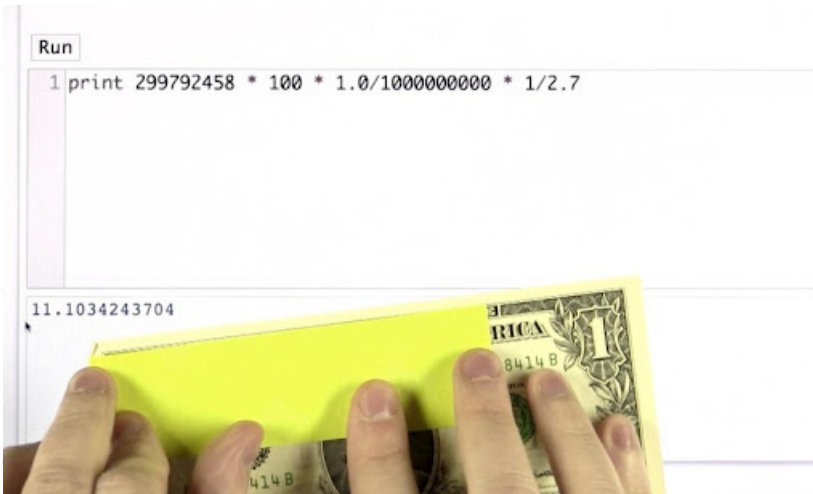
The reason for computing this is because the distance light travels in a nanosecond really matters in computing! A typical computer today executes billions of steps every second. The processor I am using is a 2.7 GHz processor. The GHz means gigahertz which is a billion cycles per second. So, the computer executes 2700000000 cycles per second.

You can think of each cycle as executing a very small instruction step. If you are using a Mac, you can see how fast your processor is by selecting the Apple menu and choosing About this Mac. If you are using a Windows 7 machine, open the Control Panel and select System and Security, then under System select View amount of RAM and processor speed.

We can compute how far light travels in the time it takes for the computer to complete one cycle:

```
print 299792458 * 100 * 1.0/1000000000 *1/2.7
11.1034243704
```


This is approximately 3/4 of the length of a dollar bill.



A **processor** is the part of the computer that carries out the steps specified in a computer program. Sometimes people call the processor the "central processing unit" or CPU.

A processor has to be small to execute programs quickly. If your computer's processor were any larger than the size of a dollar bill, then you couldn't even send light from one end of the processor to the other before finishing the execution of a single step in a program.

[Answer](#)

Admiral Grace Hopper (1906- 1992)

[Grace Hopper](#) was a pioneer in computing who was known for walking around with nanosticks. Nanosticks are pieces of wire that are the length light travels in a nanosecond, about 30 cm.

Hopper wrote one of the first programming languages, COBOL, which was for a long time the world's most widely used programming language. Hopper built the first compiler. A compiler is a program that takes as input a program in a programming language easy for humans to write and outputs a program in another language that is easier for computers to execute. The difference between a compiler and an interpreter like Python is that a compiler does all the work at once and creates a new program, whereas, the interpreter converts the source code one step at a time as the program runs.

When Grace Hopper started building the first compiler, most people did not believe it was possible for a computer program to produce other computer programs: "Nobody believed that I had a running compiler and nobody would touch it. They told me computers could only do arithmetic."

Variables

A **variable** is a name that refers to a value. In Python, we can use any sequence of letters and numbers and underscores (`_`) when 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  
Dorina  
item73
```

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

- *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/sec:

```
print speed_of_light * 100
```

You can create new variables to keep track of values in programs. Here is an expression to find the length of the nanostick in centimeters:

```
speed_of_light = 299792458  
billionth = 1.0 / 1000000000  
nanostick = speed_of_light * billionth * 100  
print nanostick
```

Quiz 7: Variables

Given the variables defined below, write Python code that prints out the distance, in meters, that light travels in one processor cycle. We use the hash mark (#) to introduce a comment. After the hash, we can write anything we want. The rest of the line is treated as a comment. The comment is not interpreted by the Python interpreter, but it is useful for humans reading the code.

Compute this by dividing the speed of light by the number of cycles per second.

```
speed_of_light = 299792458      # meters per second  
cycles_per_second = 2700000000. # 2.7 GHz
```

[Answer](#)

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.

For example, you can change the value of **cycles_per_second**. Suppose you have a faster processor:

```
speed_of_light = 299792458# meters per second
cycles_per_second = 27000000000.# 2.7GHz
cycle_distance = speed_of_light / cycles_per_second
cycle_per_second = 2800000000.# 2.8 GHz
print cycle_distance
0.111034243704

cycle_distance = speed_of_light/ cycles_per_second
print cycle_distance
0.107068735
```

Since the value that a variable refers to can change, the same exact expression can have different values at the different times it is executed.

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, $\hat{=}$, showing that the value the right side evaluates to is being assigned to the variable name on the left side.

Quiz 8: Varying Variables Quiz 1

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

[Answer](#)

Quiz 9: Varying Variables Quiz 2

What is the value of seconds after running this code:

```
minutes = minutes + 1
seconds = minutes * 60
```

- a. 0
- b. 60
- c. 120
- d. Error

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

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

[Answer](#)

Quiz 10: Spirit Age

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
```

[Answer](#)

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 **NameError**.

Quiz 11: Valid Strings

Which of the following are valid strings in Python?

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

[Answer](#)

Ada

Augusta Ada King, Countess of Lovelace, 1815-1852, was arguably the world's first computer programmer. Grace Hopper wasn't the first person to think about using computers to do things other than arithmetic, but Ada probably was. In her 1843 notes on programming the Analytical Engine, she writes:

It might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine...

The "it" King is referring to is the Analytical Engine, a mechanical programmable computer that Charles Babbage designed. Although he did not succeed in building it in the 1840s, he did have a design for it, which Ada was thinking about programming to compose music.

Quiz 12: Hello!!!

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

Strings and Numbers

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!

[Answer](#)

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 you have the string 'udacity' and you want to select the character in the zero position (that is, the first character), you would write:

```
'udacity'[0]
```

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

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

```
'udacity'[0]
'u'
'udacity'[1 + 1]
'a'
name = 'Dave'
name[0]
'D'
```

When you use negative numbers in the index, Python 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 **IndexError** indicating that the index is out of range:

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

Quiz 13: Same Value

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]`

[Answer](#)

Selecting Sub-Sequences

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 you consider the string 'udacity,' that has 7 characters, there are 7 positions: 0,1...6, with 'u' being in the 0 position.

- `<string>[<expression>]` â†’ a one-character string
- `<string>[<start expression>:<stop expression>]` â†’ 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'
```

MY NOTES: INDEXING BLANK STRINGS

When indexing a string that is "Blank" (i.e. string = ""), the following rules apply: where # is any number

- print string [#] --> would yield an ERROR
- print string [:] --> would yield "" NOT an error!
- print string [#:#] --> would yield "" NOT an error!
- print string [-#:-#] --> would yield "" NOT an error!
- print string [#:-#] --> would yield "" NOT an error!
- print string [0:0] --> would yield "" NOT an error!

Quiz 14: Capital Udacity

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

```
s = 'audacity'
```

[Answer](#)

Quiz 15: Understanding Selection

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:]`

Answer

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 16: Testing

Which of the following evaluate to `-1`:

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

[Answer](#)

Quiz 17: Testing 2

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`

[Answer](#)

Find with Numbers

In addition to passing in a target string to `find`, you 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 input position. If there is no occurrence at or after that position, the output is `-1`.

For example:

```
danton = "De l'audace, encore de l'audace, toujours de l'audace."
print danton.find('audace')
5
print danton.find('audace', 0)
5
print danton.find('audace', 5)
5
print danton.find('audace', 6)
25
print danton.find('audace', 25)
25
print danton.find('audace', 48)
-1
```

Quiz 18: Find with Numbers

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:])`

see [this forum post (most popular response)][2] for a useful explanation.

Answer

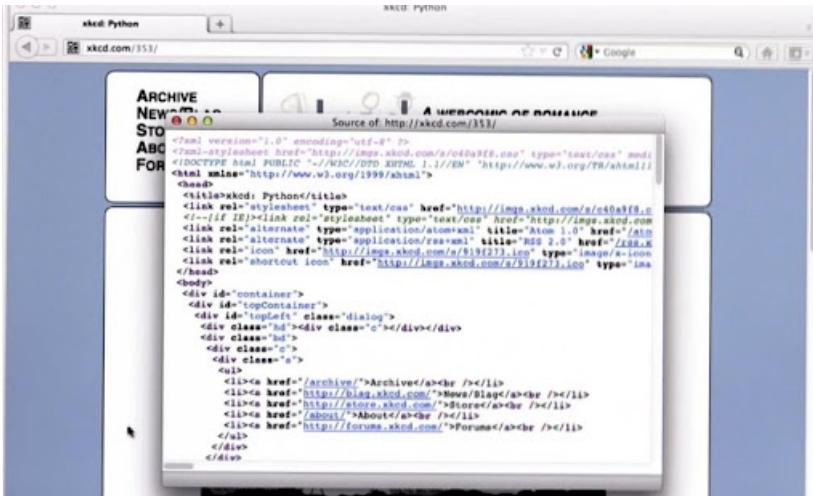
Extracting Links

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:



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 19: Extracting Links

Write Python code that initializes the variable `start_link` to be the value of the position where the first `'http://udacity.com'`.

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

[ your code here ]''

print url
http://udacity.com
```

Answer

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.

Answer Key

QUIZ 1: ANSWER

a, b, and c. We will look at ranking web pages in Unit 6.

QUIZ 2: ANSWER

All are true, but e is the best answer. In particular, it is important that programs can be read and understood by humans, even if they will **also** be executed by computers.

QUIZ 3: ANSWER

```
print 7 * 7 * 24 * 60
70560
```

QUIZ 4: ANSWER

a & b, follow the rules

QUIZ 5: ANSWER

1. valid
2. not valid, doesn't follow rule, expression --> (expression), b/c have to have () on right and left
3. valid
4. not valid doesn't follow expression a Expression Operator Expression
5. valid, nested parentheses valid but unnecessary

QUIZ 6: ANSWER

```
print 299792458 * 100 * 1/1000000000
29
```

Note that with division, Python still reads the result as an integer and the result is truncated to the resulting whole number, not rounded. For a better result, we should make one of the numbers a floating point number by adding a decimal point. Here, we changed 1 to 1.0:

```
print 299792458 * 100 * 1.0/1000000000
29.9792458
```

QUIZ 7: ANSWER

```
speed_of_light = 299792458 # meters per second
cycles_per_second = 27000000000. # 2.7GHz
```

```
print speed_of_light/ cycles_per_second  
0.111034243704
```

You could also solve this by introducing another variable:

```
speed_of_light = 299792458 # meters per second  
cycles_per_second = 27000000000. # 2.7GHz  
cycle_distance = speed_of_light/ cycles_per_second  
  
print cycle_distance  
0.111034243704
```

This is more useful since now we can easily use **cycle_distance** in other computations.

QUIZ 8: ANSWER

d

QUIZ 9: ANSWER

Error. The first assignment has minutes on the right side, but it was not yet defined since we did not yet use minutes on the left side of an assignment. The error Python gives for this is a [NameError](#):

```
Traceback (most recent call last):  
  File "/code/knownv/input/test.py", line 2, in <module>  
    minutes = minutes +1  
NameError: name 'minutes' is not defined
```

QUIZ 10: ANSWER

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

QUIZ 11: ANSWER

a and e

QUIZ 12: ANSWER

```
name = "Dave"
```

Of course, your answer will be different, unless your name is Dave too.

QUIZ 13: ANSWER

a, b and e. For c, the first expression is a two-character string, where the second is the one-character string of

the letter at position 1. For `d`, the results are the same only if the value of `s` contains at least two characters. For example, if `s = 'a'`, the first expression produces an indexing error and the second expression outputs `'i'`.

- `<string>[<expression>] -->` one-character string
- `<string>[<expression>:<expression>] -->` string that is a subsequence of the characters in the string

QUIZ 14: ANSWER

Select the part of the string we can use, from position two until the end of the string:

- `s[2:]`

Add the upper-case `U` to return the desired result:

- `'U' + s[2:]`

Use the `print` command to print it out `'Udacity'`:

```
print 'U' + s[2: ]
Udacity
```

QUIZ 15: ANSWER

a, b, c, and e. Note that e is equivalent even if `s` has fewer than three characters!

```
<string>.find(<string>)
```

QUIZ 16: ANSWER

The `find` method returns `-1` when the target string is not in the first string: c and d

QUIZ 17: ANSWER

- a. yes, no matter what string `s` is we will always find `s` at the beginning of itself
- b. no, first letter has to be `s`
- c. yes
- d. yes
- e. yes - the `s.find(s + '!!!')` part always evaluates to `-1`, since you can never find a string longer than `s` in `s`. Adding 1 to this produces 0.

QUIZ 18: ANSWER

None of the above are equivalent! This was a tricky question. Choice c is almost equivalent, except for the case where `t` is not found at all in `s[i:]`, and the output of the `find` is `-1`.

QUIZ 19: ANSWER

```
start_link = page.find('<a href=')
```

QUIZ 20: ANSWER

To extract the URL, we need to find the starting and ending double quotes and then extract the subsequence of

characters between them.

Step 1: Initialize the variable **start_quote**, which is where the double quote that starts the URL is:

```
start_quote = page.find('\"', start_link)
```

Note that it is important to pass in **start_link** as the second input to **find**, so we only find a string inside the anchor tag. Otherwise, there might be other strings on the page that are not link targets that would be found.

Step 2: Find the end quote:

```
end_quote = page.find ('\"', start_quote + 1)
```

We need to pass in **start_quote + 1** to find the next " after the start quote.

Step 3: Select the subsequence of characters from **page** between the start quote and **end_quote**, not including the quotes, and assign that to the variable **url**:

```
url = page[start_quote + 1 : end_quote]
```