Lesson #7

Web Programming with ASP, CGI/Ruby, and CGI/Python

7.1 ASP: What Is ASP?

- Active Server Pages (ASP) is Microsoft's server-side script engine for dynamically-generated web pages. It is marketed as an add-on to Internet Information Services (IIS).
- Several scripting languages may be used in ASP. However, the default scripting language (in classic ASP) is VBScript.
- We are talking here about classic ASP, not ASP.NET which is a framework.

ASP or PHP?

- Fans of either technology are usually very camped in their positions. In reality, both solutions are somewhat similar.
- PHP is based on Unix and will be your preferred choice if you are already proficient in JavaScript. ASP is based on Microsoft platforms and uses VisualBasic.
- Most choice are history-based. For example, bulletin boards are mostly written in PHP and e-commerce sites mostly in ASP.

A Simple Program

- ASP (like PHP) is included code inside the XHTML code. Its inclusion looks more like SSI than Perl.
- Like SSI and PHP, ASP programs need a special extension (.asp).
- ASP code appears between the special tags <% and %> called wrappers.

```
<%
response.write ("This is my first ASP program.")
%>
```

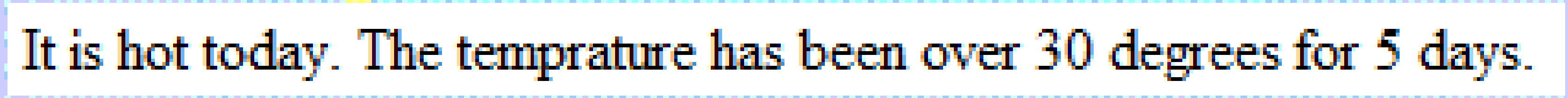
ASP Operators

- The mathematical operators in ASP are similar to many other programming languages. However, ASP does not support shortcut operators like ++, --, +=.
- Comparison operators are **=**, **<**, **>**, **<=**, **>=**, **<>**.
- Logical operators are **and**, **or**, **not**.
- The only string operator is the string concatenation operator "&" that takes two strings and slams them together to form a new string.

Using Variables

- In ASP, there is no ; after each statement. No $ for variable names (unlike Perl or PHP).
- Elements inside response.write are separated by &.

```
<%
days = 5
sentence = "It is hot today."
temp = 30.0
response.write (sentence & " The temprature has
   been over " & temp & " degrees for " & days & "
   days.")
%>
```



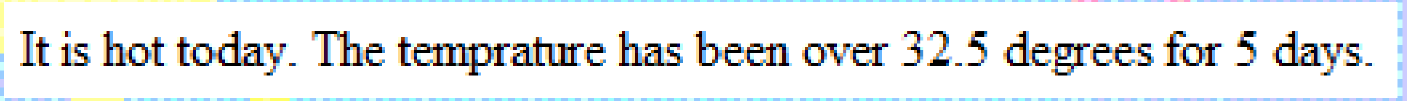
```
It is hot today. The temprature has been over 30 degrees for 5 days.
```

Using Variables

- Once again, ASP uses VBScript by default and so it also uses VBScripts variable naming conventions. These rules are:
- 1. Variable name must start with an alphabetic character (A - Z or a - z).
- 2. Variables cannot contain a period.
- 3. Variables cannot be longer than 255 characters .
- 4. Variables must be unique in the scope in which it is declared.

Using Variables inside HTML

```
<%
days = 5
sentence = "It is hot today."
temp = 32.5
%>
<% response.write (sentence) %> The temprature has
   been over <% response.write (temp) %> degrees
   for <% response.write (days) %> days.
```



```
It is hot today. The temprature has been over 32.5 degrees for 5 days.
```

ASP - Selection (if)

Note: equality in ASP is =, not ==. Not equal is <> not !=

No brackets!

```
<%
distance = 850
if distance <= 500 then
     response.write ("Easy in one day.")
else
     if distance <= 800 then
          response.write ("Feasible in one day.")
     else
          response.write ("I should reserve a
   hotel room.")
     end if
end if
%>
```

ASP - Select Case

- ASP uses the Select Case statement to check for multiple equality conditions of a single variable. The Select statement resembles a switch statement that other programming languages use.

```
<%
number = 3
select case number
  case 2
  Response.Write("number is Two")
  case 3
  Response.Write("number is Three")
  case else
  Response.Write("number is " & number)
end select
%>
```

ASP - Select Case

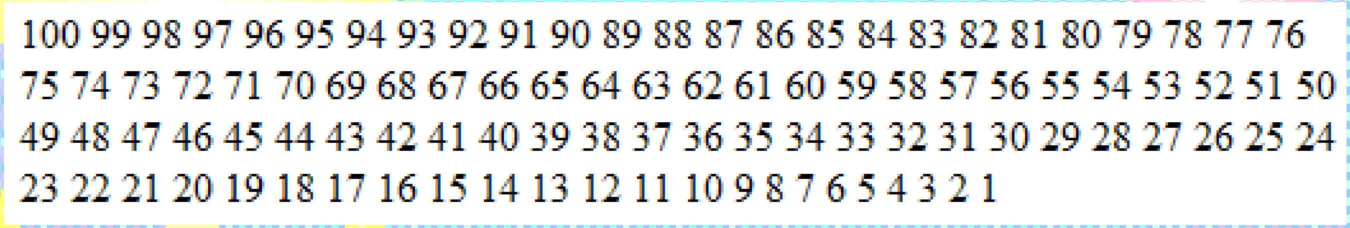
- **Select case** can use strings as well as integers for control variable.

```
<%
pet = "cat"
select case pet
  case "dog"
    Response.Write("I own a dog")
  case "cat"
    Response.Write("I own a cat")
  case Else
    Response.Write("I do not have a pet")
end select
%>
```

ASP - Loops

```
<%
count = 100
do while count <> 0
   response.write (count & " ")
   count = count - 1
loop
%>
```

--count not supported in ASP!



```
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76
75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50
49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

ASP - Arrays

- Arrays in ASP follow the exact same form and rules as those arrays in VBScript. You can create an array of specific size or you can create a dynamic sized array. Below we have examples of both types of arrays.

```
<%
Dim myFixedArray(3)  'Fixed size array
Dim myDynArray()  'Dynamic size array
%>
```

Notice the comments syntax!

ASP - Fixed Arrays

```
<%
Dim myFixedArray(3) 'Fixed size array
myFixedArray(0) = "Albert Einstein"
myFixedArray(1) = "Carl Sagan"
myFixedArray(2) = "Michio Kaku"
myFixedArray(3) = "Louis de Broglie"
For Each item In myFixedArray
        Response.Write(item & "<br />")
Next
%>
```

Size is maximum index in ASP!

Another type of loop!

ASP - Dynamic Arrays

- To create an array whose size can be changed at any time simply do not put a number within the parenthesis when you declare the array. When you know what size you want the array to be use the ReDim keyword. You may ReDim as many times as you wish.
- If you want to keep your data that already exists in the array then use the Preserve keyword.

```
<%

Dim myDynArray()  'Dynamic size array

ReDim myDynArray(1)

myDynArray(0) = "Albert Einstein"

myDynArray(1) = "Carl Sagan"

ReDim Preserve myDynArray(3)

myDynArray(2) = "Michio Kaku"

myDynArray(3) = "Louis de Broglie"

%>
```

ASP - Sessions

- The Session Object in ASP allows you to keep information specific to each of your site's visitors. Information like username, shopping cart, and location can be stored for the life of the session. The most important thing to know about ASP's Session Object is that it is only created when you store information into the Session Contents collection.
- To store a Session Variable you must put it into the Contents collection. Here we are saving the Time when someone visited the page into the Session Contents collection and then displaying it .

key

value

```
<%
Session("TimeVisited") = Time()
Response.Write("You visited this site at: " &
   Session("TimeVisited"))
%>
```

ASP - Sessions

- The ASP Session ID is the unique identifier that is automatically created when a Session starts for a given visitor. The Session ID is a property of the Session Object and is rightly called the SessionID property.
- A Session will not last forever, so eventually the data stored within the Session will be lost. There are many reasons for a Session being destroyed. The user could close their browser or they could leave their computer for an extended amount of time and the Session would time out. You can set how long it takes, in minutes, for a session to time out with the Timeout property.

```
<%
mySessionID = Session.SessionID
Session.Timeout = 240
%>
```

ASP - Cookies

- Creating an ASP cookie is exactly the same process as creating an ASP Session. Once again, you must create a key/value pair where the key will be the name of our "created cookie". The created cookie will store the value which contains the actual data.

```
<%
Response.Cookies("language") = "Russian"
%>
```

- To retrieve the cookie we use the Request.Cookies method.

```
<%
favlang = Request.Cookies("language")
Response.Write("You set the language to " &
   favlang)
%>
```

ASP - Cookies

- In ASP you can set how long you want your cookies to stay fresh and reside on the user's computer. A cookie's expiration can hold a date; this date will specify when the cookie will be destroyed.

```
<%
'create a 10-day cookie
Response.Cookies("language") = "Persian"
Response.Cookies("language").Expires = Date() + 10

'create a static date cookie
Response.Cookies("name") = "Suzy Q."
Response.Cookies("name").Expires = #January
  1,2009#
%>
```

ASP - Cookie Collections

- In ASP you can store data in a collection of cookies referring to a single object.

```
<%
Response.Cookies("brownies")("numberEaten") = 13
Response.Cookies("brownies")("eater") = "George"
Response.Cookies("brownies")("weight") = 400
For Each key In Request.Cookies("Brownies")
    Response.Write("<br />" & key & " = " & _
                   Request.Cookies("Brownies")(key))
Next
%>
```

ASP - Using Forms

- Like Perl and PHP, you can use forms with an HTML part and a server-side (ASP) part.

```
<form action="process.asp" method="post">
<select name="item">
<option>Paint</option>
<option>Brushes</option>
<option>Erasers</option>
</select>
Quantity: <input name="quantity" type="text" />
<input type="submit" />
</form>
```

ASP - Using Forms

- When the POST method is used to send data you retrieve the information with the Request Object's Form collection.

```
<%
item = Request.Form("item")
quantity = Request.Form("quantity")
Response.Write("Item: " & item & "<br />")
Response.Write("Quantity: " & quantity & "<br />")
%>
```

ASP - Query Strings

- To extract data from query strings (or forms using the GET method), we use the Request Object's QueryString collection.
- For example, with a url like www.domain.com/query.asp?Name=Lucy&Age=22

```
<%
name = Request.QueryString("Name")
age = Request.QueryString("Age")
%>
```

Using JavaScript with ASP

- VBScript is the default scripting language that ASP is coded in, so if you want to specify a different scripting language you have to state which scripting language you will be using at the very beginning of your code. Below is the line of code that must be your first line of ASP code or else your page will break and you'll get an error message.

```
<%@ Language="javascript"
'The rest of your ASP Code....
%>
```

Using JavaScript with ASP

- VBScript is the default scripting language that ASP is coded in, so if you want to specify a different scripting language you have to state which scripting language you will be using at the very beginning of your code. Below is the line of code that must be your first line of ASP code or else your page will break and you'll get an error message.

```
<%@ Language="javascript"
'The rest of your ASP Code....
%>
```

7.2 Ruby: What Is Ruby?

- Ruby is a reflective, dynamic, object-oriented programming language. It combines syntax inspired by Perl with Smalltalk-like object-oriented features.
- Ruby is a single-pass interpreted language. Its official implementation is free software written in C.

Ruby

- Ruby originated in Japan during the mid-1990s and was initially developed and designed by Yukihiro "Matz" Matsumoto. It is based on Perl, Smalltalk, Eiffel, Ada, and Lisp.
- Ruby is a general-purpose language. It is not properly a web language. Even so, web applications and web tools in general are among the most common uses of Ruby.
- Not only can you write your own SMTP server, FTP daemon, or Web server in Ruby, but you can also use Ruby for more usual tasks such as CGI programming or as a replacement for PHP.

Ruby Syntax

- The syntax of Ruby is broadly similar to Perl. Class and method definitions are signaled by keywords. In contrast to Perl, variables are not obligatorily prefixed with a sigil (like $). When used, the sigil changes the semantics of scope of the variable.
- In Ruby, ordinary variables lack sigils, but "$" is prefixed to global variables, "@" is prefixed to instance variables, and "@@" is prefixed to class variables.

Ruby Syntax

- The most striking difference from C and Perl is that keywords are typically used to define logical code blocks, without braces. Line breaks are significant and taken as the end of a statement; a semicolon may be equivalently used.

Ruby Syntax

- Comments start with a pound/sharp (#) character and go to the end of line.
- Ruby programs are sequence of expressions.
- Each expression is delimited by semicolons(;) or newlines unless obviously incomplete (for example: a trailing + or any operator).
- A backslash (\) at the end of line indicates that the expression continues on the next line.

CGI/Ruby

- Whitespace characters such as spaces and tabs are generally ignored in Ruby code, except when they appear in strings. Sometimes, however, they are used to interpret ambiguous statements. Interpretations of this sort produce warnings when the -w option is enabled (#!/usr/bin/ruby -w shebang line).
- a + b is interpreted as a+b (**a** is a local variable)
- a +b is interpreted as a(+b) (**a** is a method call)

Ruby Identifiers

- Identifiers are names of variables, constants, and methods. Ruby identifiers are case sensitive. Ruby identifier names may consist of alphanumeric characters and the underscore character.
- Reserved words may not be used as constant or variable names. They can, however, be used as method names.

Reserved Words

BEGIN	do	next	then
END	else	nill	true
alias	elsif	not	undef
and	end	or	unless
begin	ensure	redo	until
break	false	rescue	when
case	for	retry	while
class	if	return	
def	in	self	__FILE__
defined?	module	super	__LINE__

Here Document

- Here Document refers to build string`s from multiple lines. Following a << you can specify a string or an identifier to terminate the string literal, and all lines following the current line up to the terminator are the value of the string.
- Notice there must be no space between << and the terminator.

Here Document

```
#!/usr/bin/ruby
puts "Content-type: text/html\n\n"

print <<HERE1
<p style="color:red">
If you come to a fork in the road, take it.
</p>
HERE1

print <<HERE2
<p style="color:blue;">
It ain't over till it's over.
</p>
HERE2
```

If you come to a fork in the road, take it.

It ain't over till it's over.

Comments in Ruby

```
# Single line comment.
# Another single line comment.

=begin
This is a comment spread
over multiple
lines.
=end
```

Ruby Types

- Basic types are numbers, strings, ranges, symbols, arrays, and hashes. Also included are files.
- They are a combination of types known in Perl and C and they behave pretty much the same.
- For numbers, underscore can be used instead of a comma 1,000 => 1_000.

Writing Programs in CGI/Ruby

```
#!/usr/local/bin/ruby -w
puts "Content-type: text/html\n\n"
puts "Hello World!\n"
n = 2 + 4 + 6 + 8
n /= 5
m = 2 * 3 * 4
m -= n
puts "The answer is: " + m.to_s
```

Note: CGI/Ruby works like CGI/Perl in autonomous programs. Not embedded like PHP or ASP. Some servers support rhtml which is an embedded version of Ruby.

Ruby Strings

- "this is a string".
- "hello" + "world" => "helloworld"
- "ho " * 3 => "ho ho ho"
- "hello".capitalize => "Hello"
- "hello".reverse => "olleh"
- "hello".next => "hellp"
- "hello".upcase => "HELLO"
- "hello".length => 5
- Other methods include downcase, swapcase and length.

Classes and Objects

- An object is a unit of data. A class is what kind of data it is. The main classes are Integer, Float and String.
- Division (/) doesn't work the same with integers and floats.
- Addition (+) doesn't work the same with strings as it does with integers.
- Strings have several methods that integers and floats don't have (e.g., capitalize, length, and upcase).

Converting between classes

- Some examples of conversions:
- `"hello".to_i => 0`
- `"hello".to_f =>0.0`
- `3.5.to_i  => 3`
- `3.5.to_s =>  "3.5"`
- `4.to_f => 4.0`
- `5.to_s => "5"`
- You can verify the class of an object:
- `12.is_a?(Integer) => true`

Constants and Variables

- Variable names must begin with a lower-case letter.
- Constants are like variables, except that you are telling Ruby that their value is supposed to remain fixed. If you try to change the value of a constant, Ruby will give you a warning (but just a warning!).
- You define constants just like variables, except that the first letter is uppercase.

String Literals

- Double-quoted strings allow substitution and backslash notation but single-quoted strings don't allow substitution and allow backslash notation only for \\ and \'
- You can substitute the value of any Ruby expression into a string using the sequence #{ expr }. Here expr could be any ruby expression.

```
puts "The number of seconds in a day is :
  #{24*60*60}.";
puts 'escape using "\\"';
puts 'That\'s right!';
```

Ruby Operators

- Arithmetic Operators: + - * / % **
- Parallel assignments: a, b, c = 10, 20, 30
- Bitwise operators: &(and), |(or), ^(xor), ~(1s complement), <<(left shift), >>(right shift)
- Logical operators: &&, ||, ! (and, or, not also accepted).
- Range operators: 1..10 (1 to 10), 1...10 (1 to 9).

Counting Loops

```
#!/usr/local/bin/ruby -w
puts "Content-type: text/html\n\n"

4.times do
puts "Hello World! "
end
puts "<br /><br />"
count = 0
5.times do
count = count + 1
puts "Count= " + count.to_s
end
```

Conditionals

```
==   equal
!=   not equal to
>   greater than
<   less than
>= greater than or equal to
<= less than or equal to
<=> 1 if larger, -1 if smaller, 0
  if equal
```

Selection (if...else / unless)

Note that Ruby uses elsif, not else if.

unless can also be used in a similar fashion.

```
temp = 24
humid = 85
if temp < 20
   puts "It is cool outside. "
elsif temp > 26 and humid > 50
   puts "It is a muggy day. "
else
   puts "It is a comfortable day. "
end
print "It is very humid! " if humid > 80
```

Selection (case)

```
age =  5
case age
when 0 .. 2
    puts "baby"
when 3 .. 6
    puts "little child"
when 7 .. 12
    puts "child"
when 13 .. 18
    puts "youth"
else
    puts "adult"
end
```

While loops

```
#!/usr/local/bin/ruby -w
puts "Content-type: text/html\n\n"

puts "<html><body style=\"font-
size:2.5em;color:blue;\">"

count = 1
while count <= 25
puts count.to_s + " "
count = count + 1
end

puts "</body></html>";
```

While loops

```
#Calculate the highest power of two less than 10000

number = 1
while number < 10000
number = number * 2
end

number = number / 2

puts number.to_s + " is the highest " +\
"power of 2 less than 10000"
```

To spread statements on multiple lines

Sorting Arrays

```
primes = [11,5,7,2,13,3]
puts primes
puts "<br>"
puts primes.sort
puts "<br><br>"

parks = ["Yosemite", "Grand Canyon",
"Zion", "Mesa Verde", "Petrified
Forest"]
puts parks.sort
```

Operations on Arrays

```
#reverse array
puts parks.reverse

#number of elements in array
puts parks.length

#adding an element
parks = parks + ["Rocky Mountain"]

#removing elements
parks = parks - ["Yosemite", "Zion"]

#array to string
puts parks.to_s
```

Arrays and Iterators

```
islands = ["Maui", "Kaua'i", "O'ahu",
"Moloka'i", "Lana'i", "Hawai'i"]
islands.each do |island_name|
puts "The island of " + island_name +
"<br />"
end

islands.sort.each do |island_name|
puts "I am planning my vacation to " +
island_name + "<br />"
end
```

Hashes

- Hashes are the same as in Perl. They are pairs of keys and values.

```
# initial definition
cartoon = {
  "character"   => "Popeye",
  "producer"    => "Fleischer Studios",
  "year_begin"  => "1933",
  "year_end"    => "1942"
}

# adding a property
cartoon["number_of_cartoons"] = "109"
```

Iterations and Hashes

```
# printing all keys and values
cartoon.each do |key, value|
  puts key + ": " + value + "<br>"
end
```

```
character: Popeye
number_of_cartoons: 109
year_end: 1942
year_begin: 1933
producer: Fleischer Studios
```

Functions

```
#!/usr/local/bin/ruby -w
puts "Content-type: text/html\n\n"

def say_hi
puts "<div style=\"font-
size:3em;color:red;\">Hello, how are
you?</div>"
end

say_hi
say_hi
say_hi
```

Functions with arguments

```
#!/usr/local/bin/ruby -w
print "Content-type: text/html\n\n"

def say_hi (name)
puts "<div>Hello, " + name + " how are you?</div>"
end

def sum(a,b)
a + b
end


say_hi ("Fred")
say_hi ("Wilma")
x = "Barney"
say_hi (x)

puts sum(10,20)
puts sum(55,22)
```

Note: Parentheses around arguments are optional.

The yield statement

The yield statement is used to invoke a block of code.

```
def twice
yield yield
end

def vacation
yield ("Honolulu")
yield ("Tahiti")
yield ("Jamaica")
end

twice {puts "Good morning!"}

vacation do |x|
puts "<div>I'd rather be in " + x +
".</div>"
end
```

Using forms with Ruby/CGI

- Using the CGI class gives you access to the HTML query string parameters. Ruby will take care of GET and POST methods automatically.

```
#!/usr/local/bin/ruby -w
print "Content-type: text/html\n\n"
require 'cgi'
cgi = CGI.new
puts cgi['FirstName']
puts cgi['LastName']
```

Creating forms with Ruby/CGI

- CGI contains a huge number of methods used to create HTML. You will find one method per tag. In order to enable these methods, you must create a CGI object by calling CGI.new.
- Visit www.ihypress.net/programming/ruby/exec.php?script=20 for an example of a form created with the HTML methods.

Date and Time

- We can use Time object to get various components of date and time.

```
time = Time.new # or Time.now
puts "Current Time : " + time.inspect
puts time.year     # Year
puts time.month    # Month(1 to 12)
puts time.day      # Day (1 to 31 )
puts time.wday     # Day of week: 0 is Sunday
puts time.yday     # Day of year (1 to 366)
puts time.hour     # 24-hour clock
puts time.min      # Minutes (0 to 59)
puts time.sec      # Seconds (0 to 59)
puts time.usec     # Microseconds
puts time.zone     # Timezone name
```

Date and Time

```
millenium = Time.local(2000, 1, 1, 0, 0)
puts "Millenium Begins : " + millenium.inspect

time = Time.new
time.utc_offset # Offset from GMT
time.isdst      # false: If no DST.
time.gmtime     # Convert back to GMT.
time.getlocal   # New Time object in local zone
time.getutc     # New Time object in GMT

#formats
puts time.strftime("%Y-%m-%d %H:%M:%S")
```

Time formats

%a	The abbreviated weekday name (Sun).
%A	The full weekday name (Sunday).
%b	The abbreviated month name (Jan).
%B	The full month name (January).
%c	The preferred local date and time representation.
%d	Day of the month (01 to 31).
%H	Hour of the day, 24-hour clock (00 to 23).
%I	Hour of the day, 12-hour clock (01 to 12).
%j	Day of the year (001 to 366).
%m	Month of the year (01 to 12).
%M	Minute of the hour (00 to 59).
%p	Meridian indicator (AM or PM).
%S	Second of the minute (00 to 60).
%U	Week number of the current year, starting with the first Sunday as the first day of the first week (00 to 53).
%W	Week number of the current year, starting with the first Monday as the first day of the first week (00 to 53).
%w	Day of the week (Sunday is 0, 0 to 6).
%x	Preferred representation for the date alone, no time.
%X	Preferred representation for the time alone, no date.
%y	Year without a century (00 to 99).
%Y	Year with century.
%Z	Time zone name.
%%	Literal % character.

Cookies in Ruby

- You can create a named cookie object and store a any textual information in it. To send it down to the browser, set a cookie header in the call to CGI.out.

One hour

```
#!/usr/bin/ruby
require "cgi"
cgi = CGI.new("html4")
mycookie1 = CGI::Cookie.new('name' => 'province0000',
                            'value' => 'Ontario',
                            'expires' => Time.now + 3600)
cgi.out('cookie' => mycookie1) do
   cgi.head{} + cgi.body{"The cookie has been set!!"}
end
```

Ruby: Reading Cookies

```
#!/usr/bin/ruby
require "cgi"
cgi = CGI.new("html4")

mycookie2 = cgi.cookies['province0000']

if mycookie2.length > 0
 cgi.out('cookie' => mycookie2) do
         cgi.head{} + cgi.body {mycookie2[0]}
end
else
 cgi.out {cgi.html {cgi.body {cgi.div {"Cookie not
   found"} } } }
end
```

See in action:

www2.scs.ryerson.ca/~dhamelin/cgi-bin/cookie1.rb
www2.scs.ryerson.ca/~dhamelin/cgi-bin/cookie2.rb

Ruby: Sessions

- A CGI::Session maintains a persistent state for Web users in a CGI environment. Sessions should be closed after use, as this ensures that their data is written out to the store. Sessions can strore strings only.
- See an example here: www.tutorialspoint.com/ruby/ruby_cgi_sessions.htm

7.3 Python: What Is Python?

- Python is a general-purpose, interpreted high-level programming language that emphasizes code readability.
- Python supports multiple programming paradigms: object-oriented, imperative and functional programming.

About Python

- Python features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl, and Tcl. Like other dynamic languages, Python is often used as a scripting language.
- Python is a general-purpose language but is often used as a scripting language for web applications.
- Among the users of Python are YouTube, the original BitTorrent client, and Pinterest. The NASA, PBS and the Wahington Post websites also use Python extensively.

Guido van Rossum
The creator of Python

Python Syntax

- The syntax of Python is based on key words and position, not punctuation. It that regard it is close to ASP (VB Script) than Perl, PHP or Ruby.
- Also, unlike Perl or Ruby, Python has usually one way to do things instead of multiple ones.

Python - Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive language.

Python - Indentation

- Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

Python - Indentation Example

```
if True:
    print "True"
else:
    print "False"
```

Good!

```
if True:
      print "Answer"
            print "True"
else:
      print "Answer"
  print "False"
```

Bad!

Python - Quotation and Comments

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines (like heredocs).
- A comment starts with # (like Perl).

Python - Variables

- No types nor declarations in Python.

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
```

- Python allows you to assign a single value to several variables simultaneously.

```
a = b = c = 1
```

Python - Strings

- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator (kind of like in Ruby).

```
str = 'Hello World!'
print str            # Prints complete string
print str[0]         # Prints first character of the
string
print str[2:5]       # Prints characters from 3rd to 6th
print str[2:]        # Prints string starting from 3rd
character
print str * 4        # Prints string four times
print str + "<br>"   # Prints concatenated string
```

Python: Basic Language References

- Read the following pages for more details about the Python language for web programming:

tutorialspoint.com/python/python_lists.htm

tutorialspoint.com/python/python_tuples.htm

tutorialspoint.com/python/python_dictionary.htm

tutorialspoint.com/python/python_date_time.htm

tutorialspoint.com/python/python_functions.htm

Python: Files

```
out = open("data.txt", "w")
out.write( "This is in file.\n")
out.close()
in = open("data.txt", "r")
str = in.read(4);
print "Read String is : ", str
out.close()
```

If number is omitted, it reads everything!

This

Python: My First Program

```
#!/usr/bin/python

print "Content-type:text/html\n\n"

print '<html><body>'

print '<div style="font-
size:3.0em;color:#cc0000;">This is
my first Python program</div>'

print '</body></html>'
```

Python - CGI Programming

Get more details here about environment variables, forms, and cookies.

tutorialspoint.com/python/python_cgi_programming.htm

End of lesson