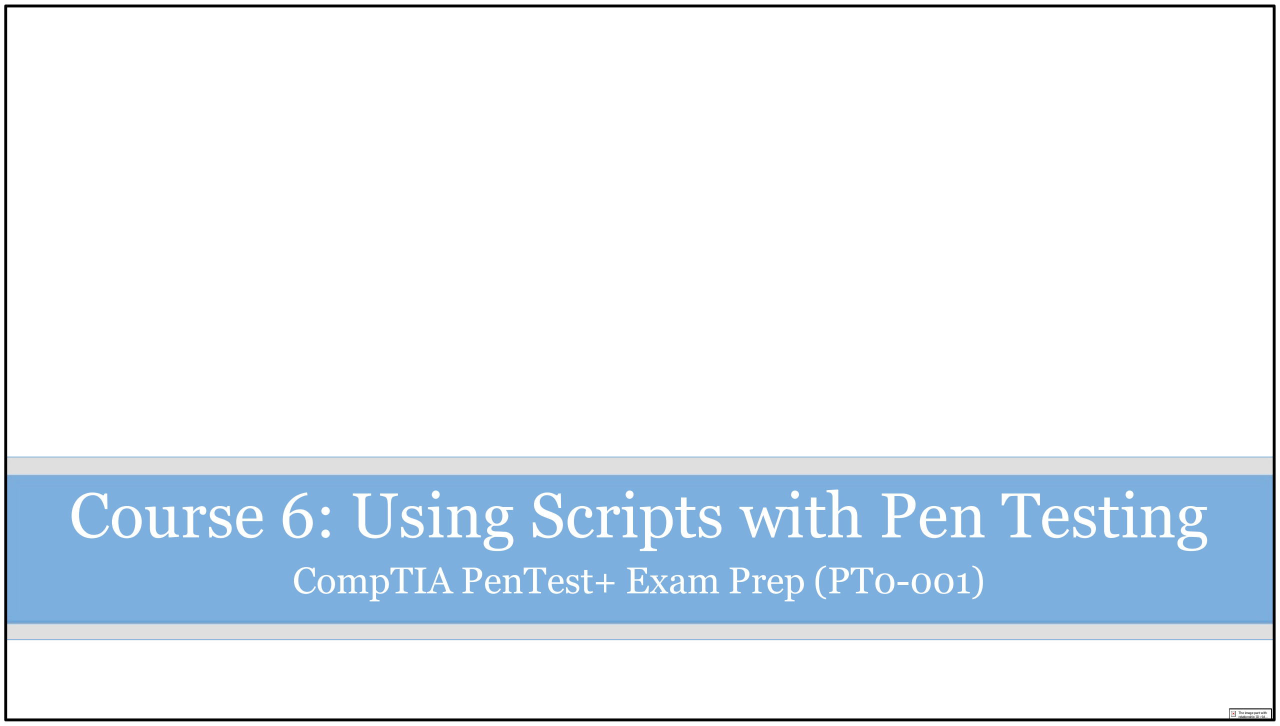
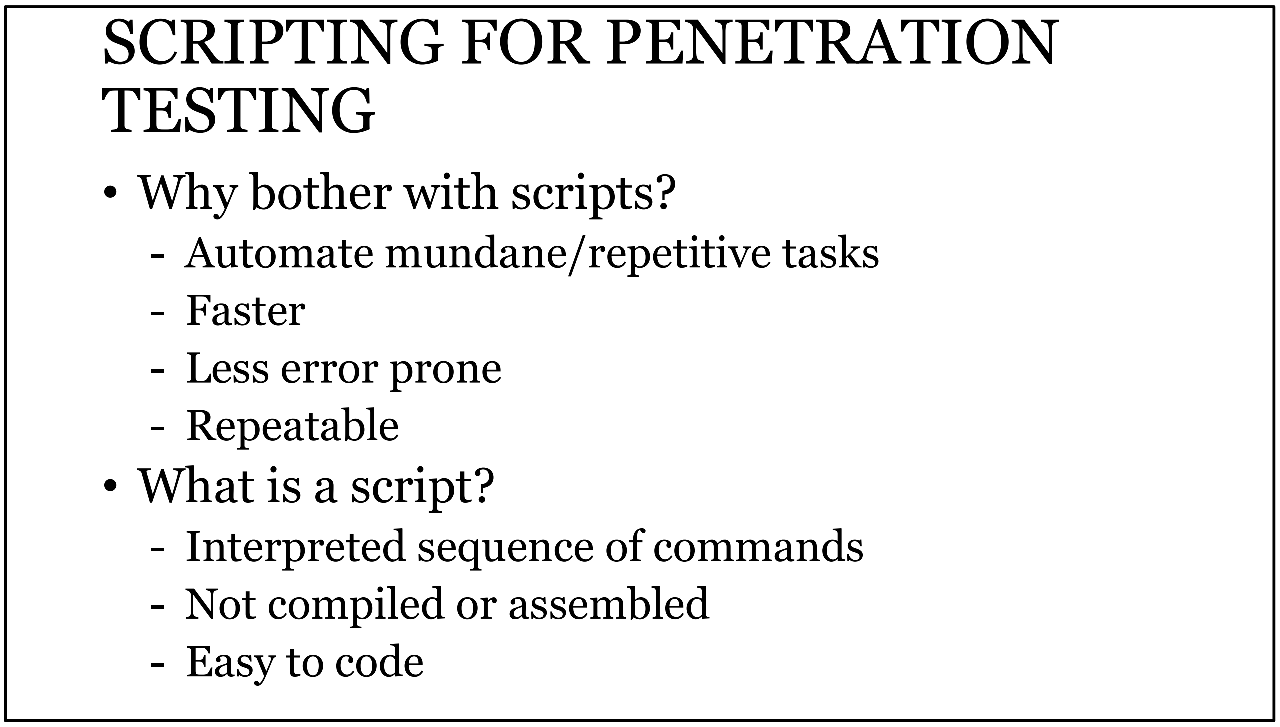


Course 6: Using Scripts with Pen Testing

CompTIA PenTest+ Exam Prep (PT0-001)

Using Scripts with Pen Testing

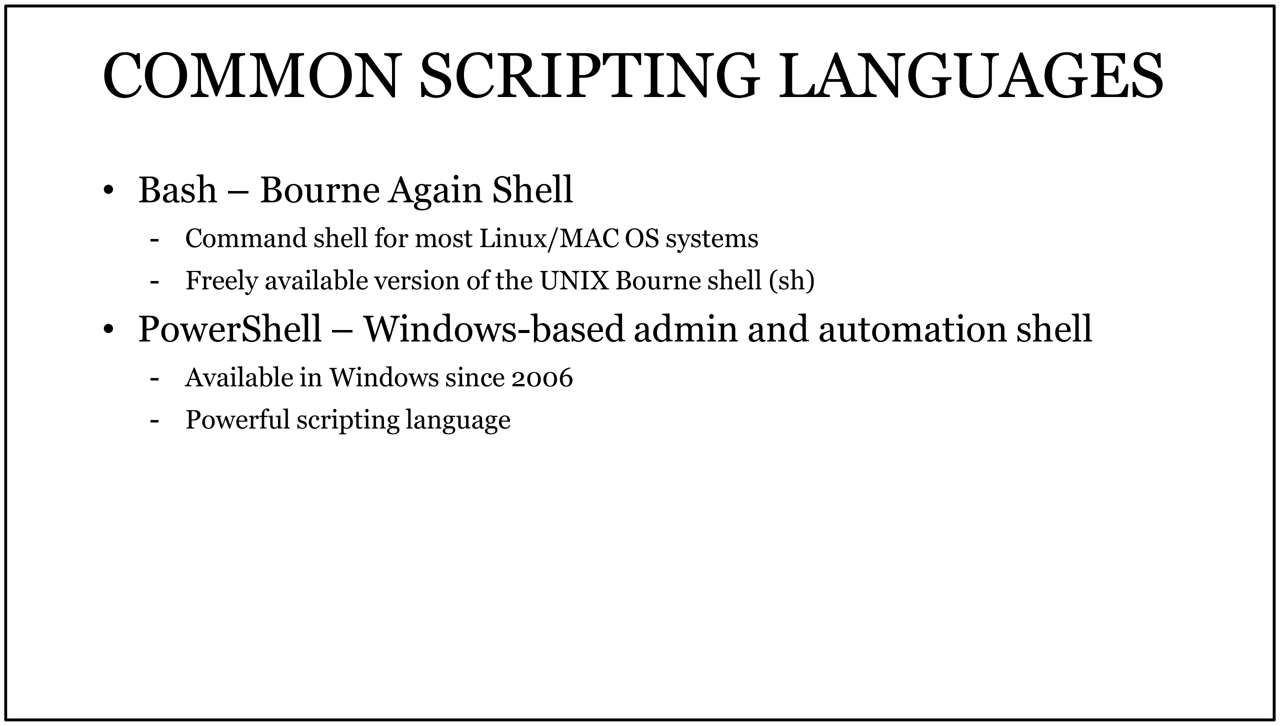
Episode 1



SCRIPTING FOR PENETRATION TESTING

- Why bother with scripts?
  - Automate mundane/repetitive tasks
  - Faster
  - Less error prone
  - Repeatable
- What is a script?
  - Interpreted sequence of commands
  - Not compiled or assembled
  - Easy to code

4.4 Given a scenario, analyze a basic script (limited to Bash, Python, Ruby, and PowerShell).



COMMON SCRIPTING LANGUAGES

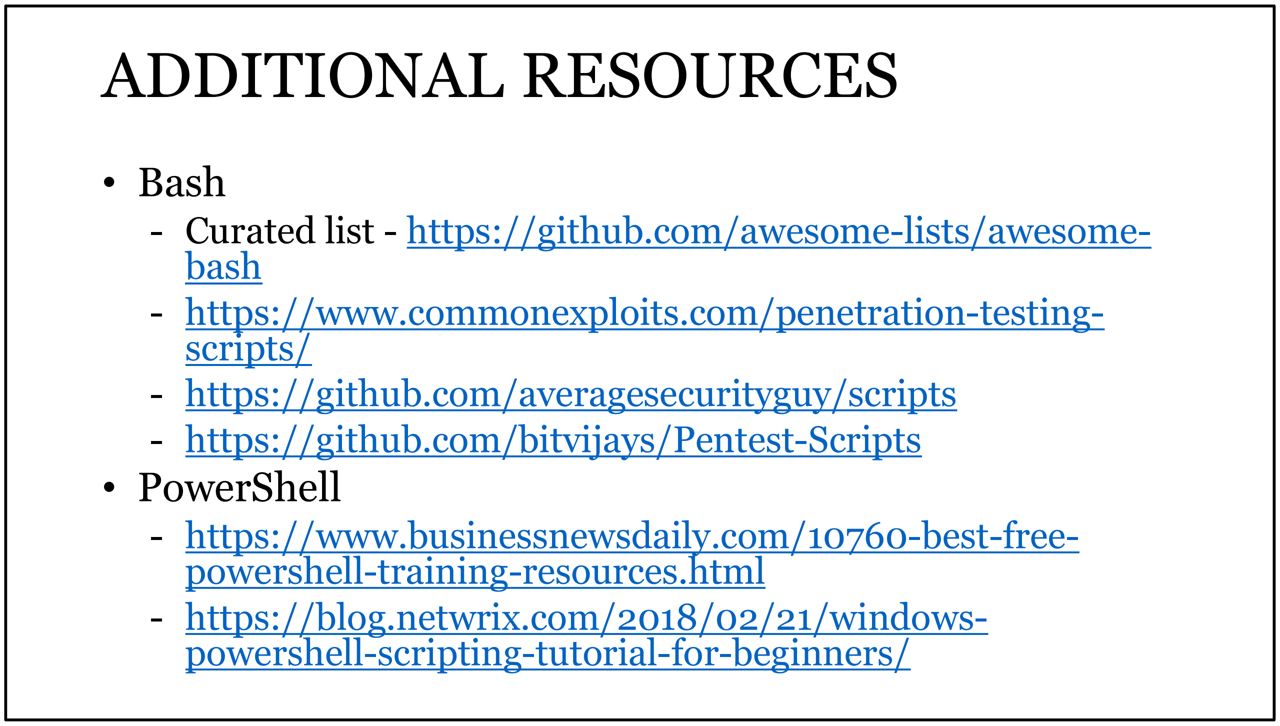
- Bash – Bourne Again Shell
  - Command shell for most Linux/MAC OS systems
  - Freely available version of the UNIX Bourne shell (sh)
- PowerShell – Windows-based admin and automation shell
  - Available in Windows since 2006
  - Powerful scripting language

4.4 Given a scenario, analyze a basic script (limited to Bash, Python, Ruby, and PowerShell).

COMMON SCRIPTING LANGUAGES

- Ruby – object-oriented high-level interpreted general purpose programming language
  - Influenced by Perl, Smalltalk, Ada, Lisp
- Python –object-oriented high-level interpreted general purpose programming language
  - Extensive available libraries
  - Great intro language

4.4 Given a scenario, analyze a basic script (limited to Bash, Python, Ruby, and PowerShell).



ADDITIONAL RESOURCES

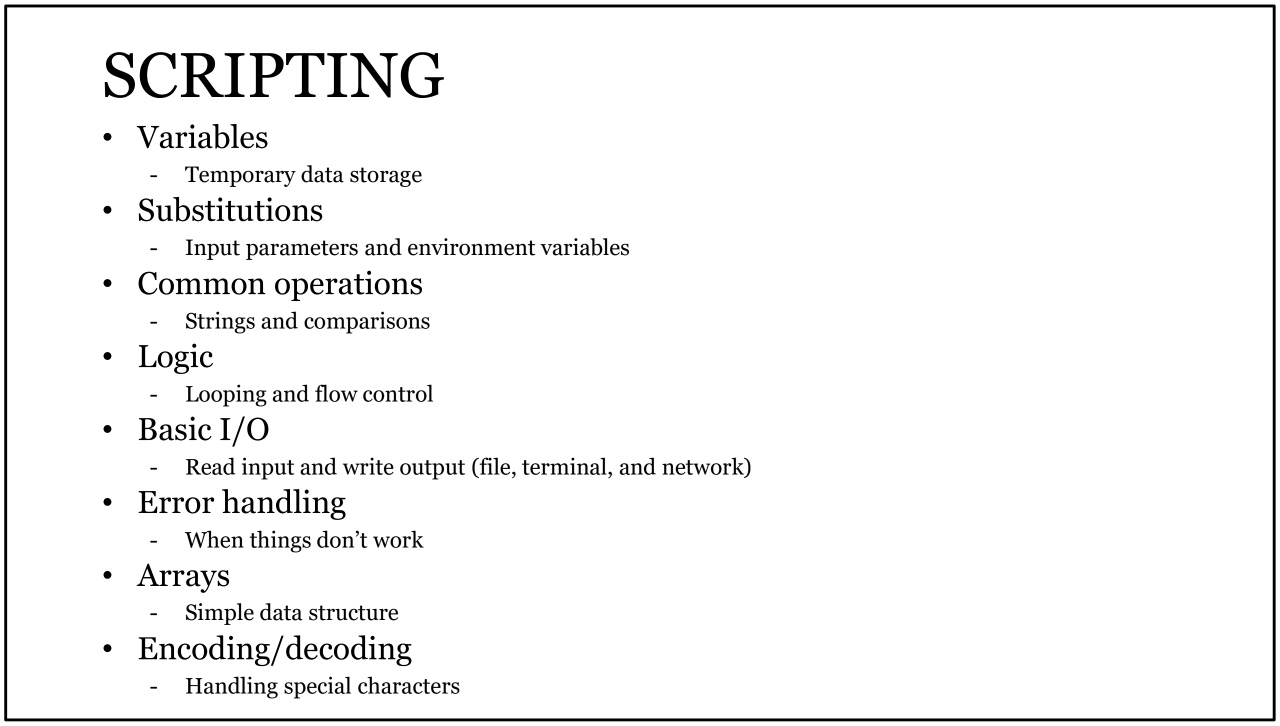
- Bash
  - Curated list - https://github.com/awesome-lists/awesome-bash
  - https://www.commonexploits.com/penetration-testing-scripts/
  - https://github.com/averagesecurityguy/scripts
  - https://github.com/bitvijays/Pentest-Scripts
- PowerShell
  - https://www.businessnewsdaily.com/10760-best-free-powershell-training-resources.html
  - https://blog.netwrix.com/2018/02/21/windows-powershell-scripting-tutorial-for-beginners/

4.4 Given a scenario, analyze a basic script (limited to Bash, Python, Ruby, and PowerShell).

ADDITIONAL RESOURCES

- Ruby
  - https://www.ruby-lang.org/en/
  - https://hackr.io/tutorials/learn-ruby
  - http://ruby-for-beginners.rubymonstas.org/index.html
- Python
  - https://learnpythonthehardway.org/
  - http://shop.oreilly.com/product/9781597499576.do

4.4 Given a scenario, analyze a basic script (limited to Bash, Python, Ruby, and PowerShell).



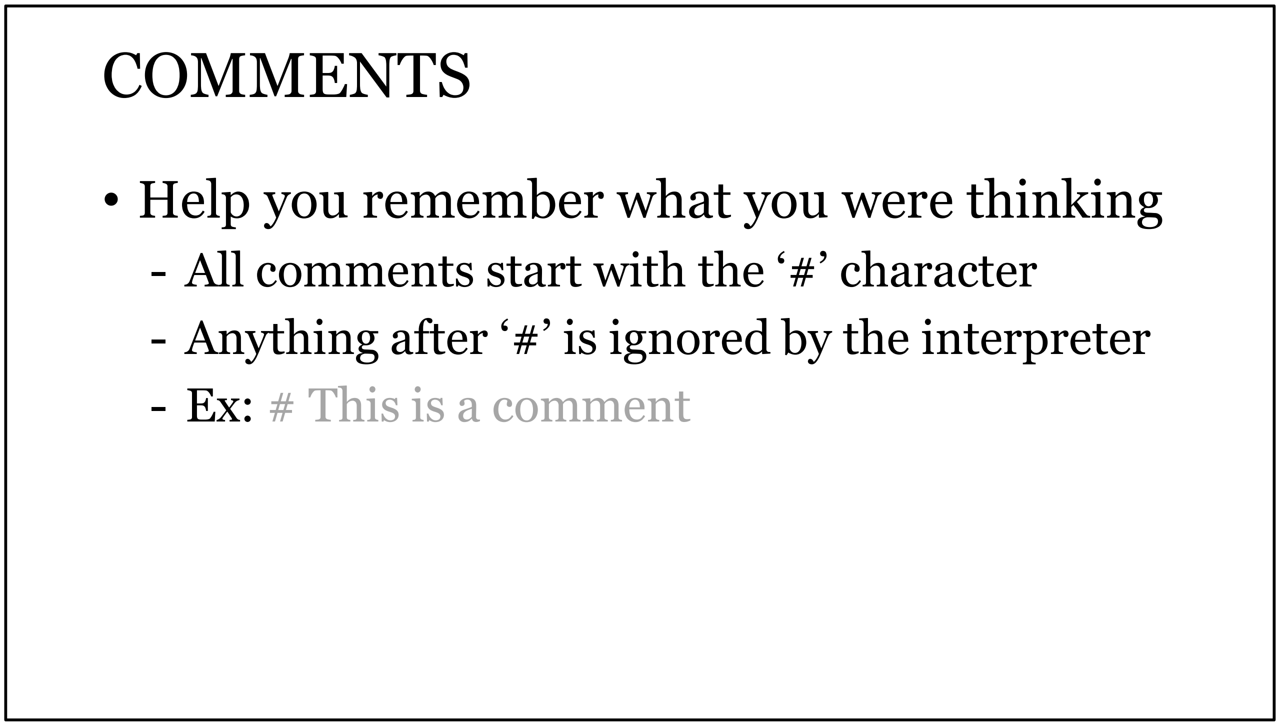
SCRIPTING

- Variables
  - Temporary data storage
- Substitutions
  - Input parameters and environment variables
- Common operations
  - Strings and comparisons
- Logic
  - Looping and flow control
- Basic I/O
  - Read input and write output (file, terminal, and network)
- Error handling
  - When things don’t work
- Arrays
  - Simple data structure
- Encoding/decoding
  - Handling special characters

4.4 Given a scenario, analyze a basic script (limited to Bash, Python, Ruby, and PowerShell).

Bash Scripting Basics

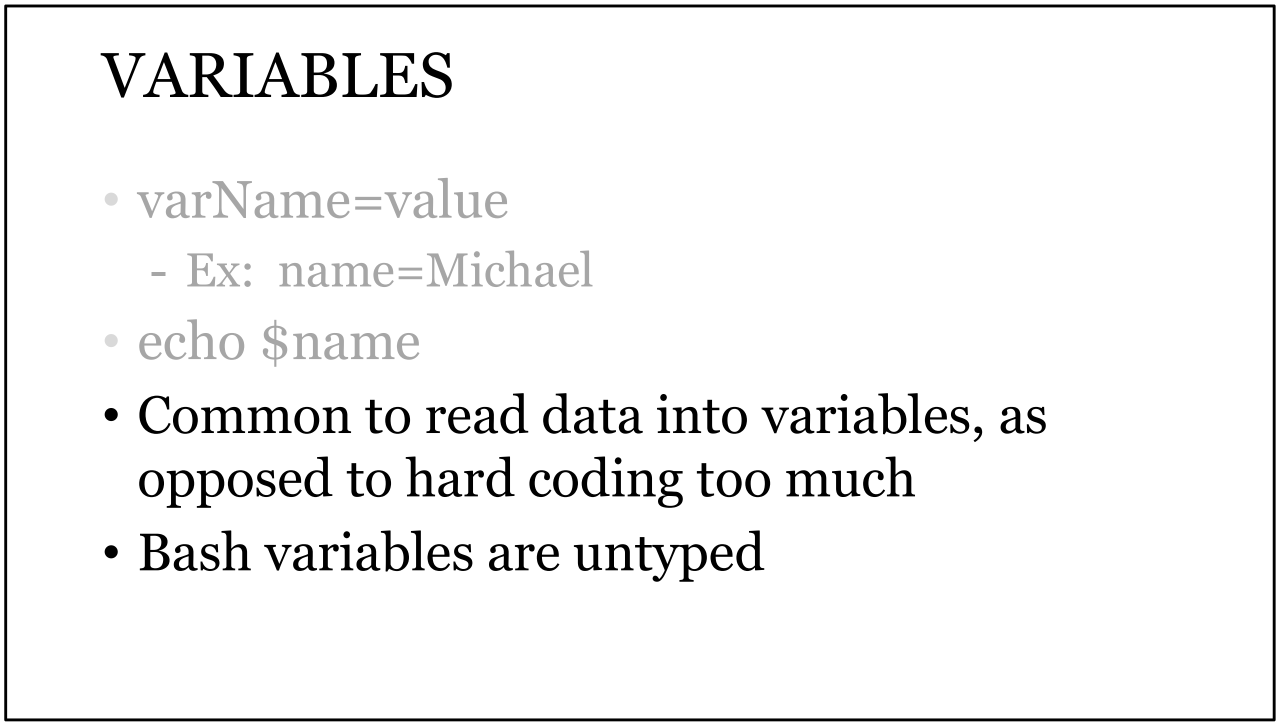
Episode 2



COMMENTS

- Help you remember what you were thinking
  - All comments start with the ‘#’ character
  - Anything after ‘#’ is ignored by the interpreter
  - Ex: # This is a comment

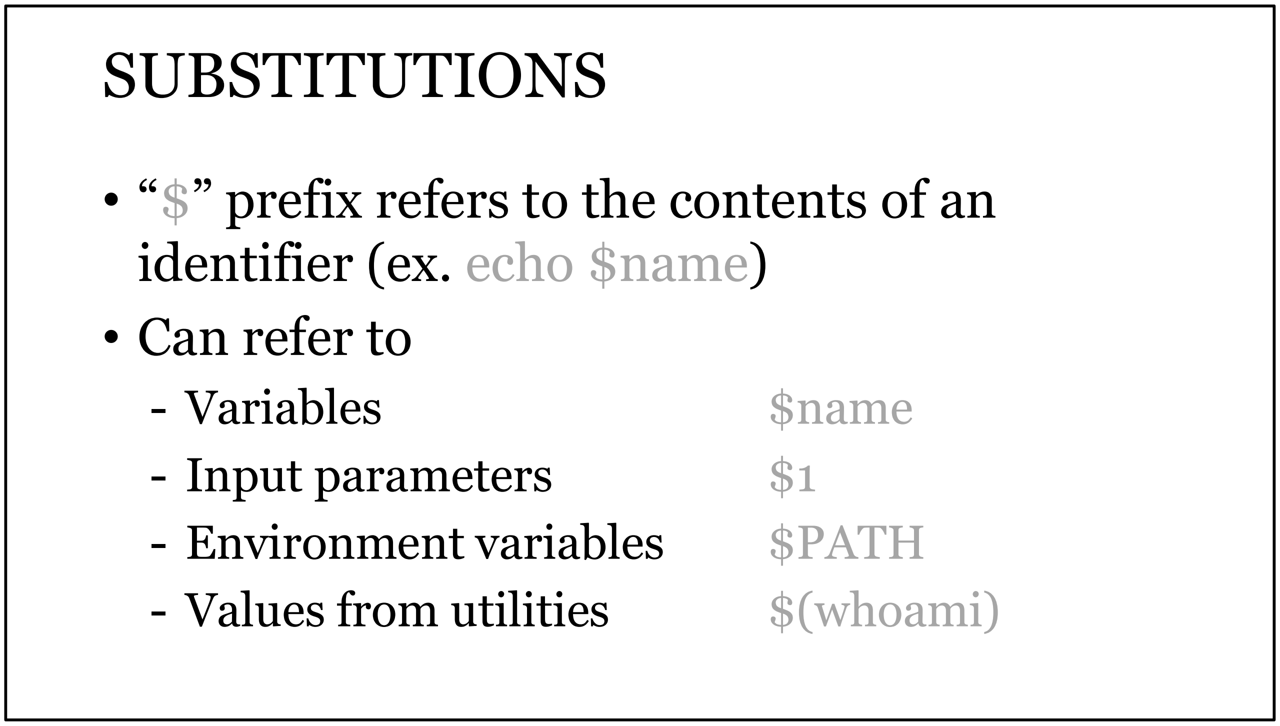
4.4.4 Bash Variables



VARIABLES

- varName=value
  - Ex: name=Michael
- echo $name
- Common to read data into variables, as opposed to hard coding too much
- Bash variables are untyped

4.4.4 Bash Variables

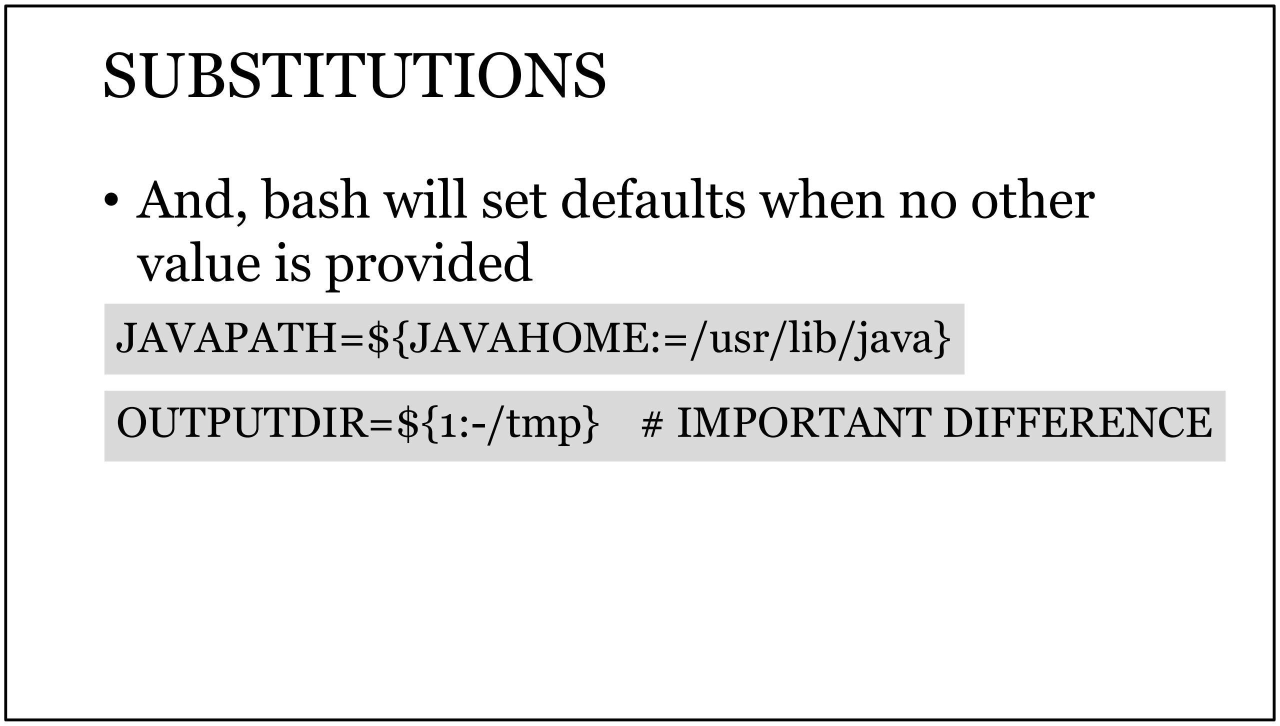


SUBSTITUTIONS

- “\$” prefix refers to the contents of an identifier (ex. echo \$name)
- Can refer to

- Variables	\$name
- Input parameters	\$1
- Environment variables	\$PATH
- Values from utilities	\$(whoami)

4.4.3 Bash Substitutions



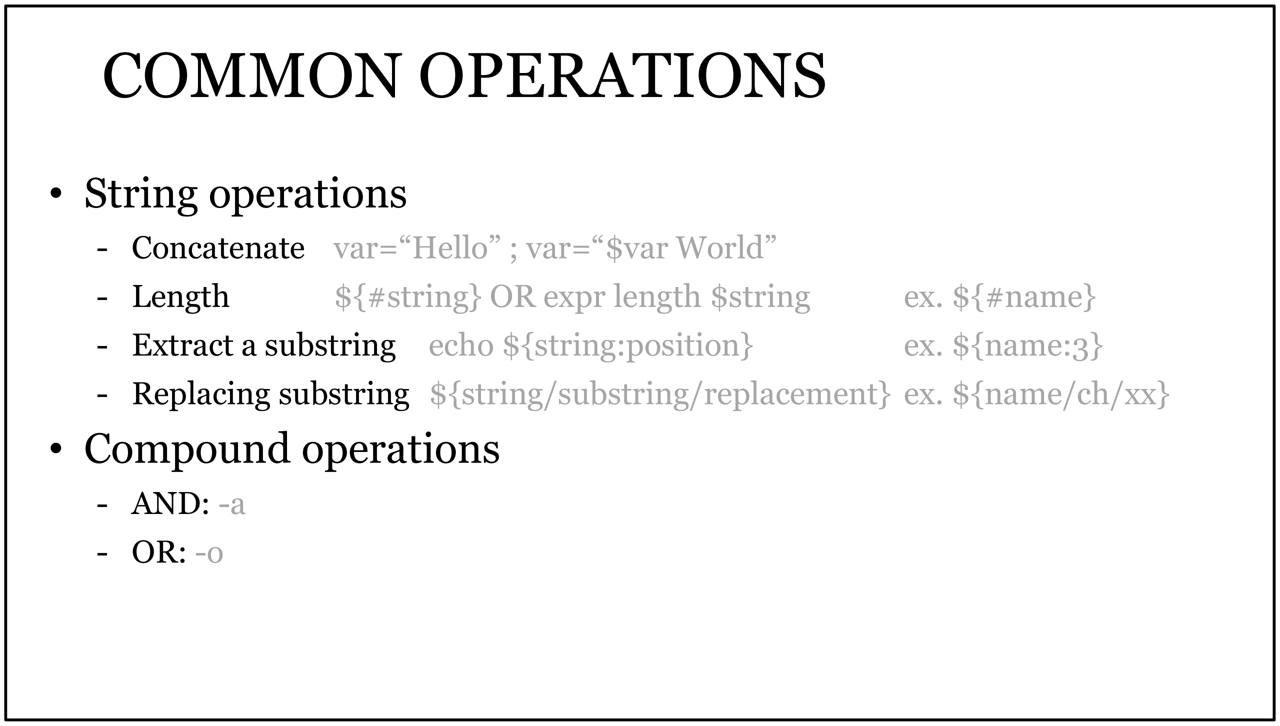
SUBSTITUTIONS

- And, bash will set defaults when no other value is provided

```
JAVAPATH=${JAVAHOME:=/usr/lib/java}
```

```
OUTPUTDIR=${1:-/tmp}   # IMPORTANT DIFFERENCE
```

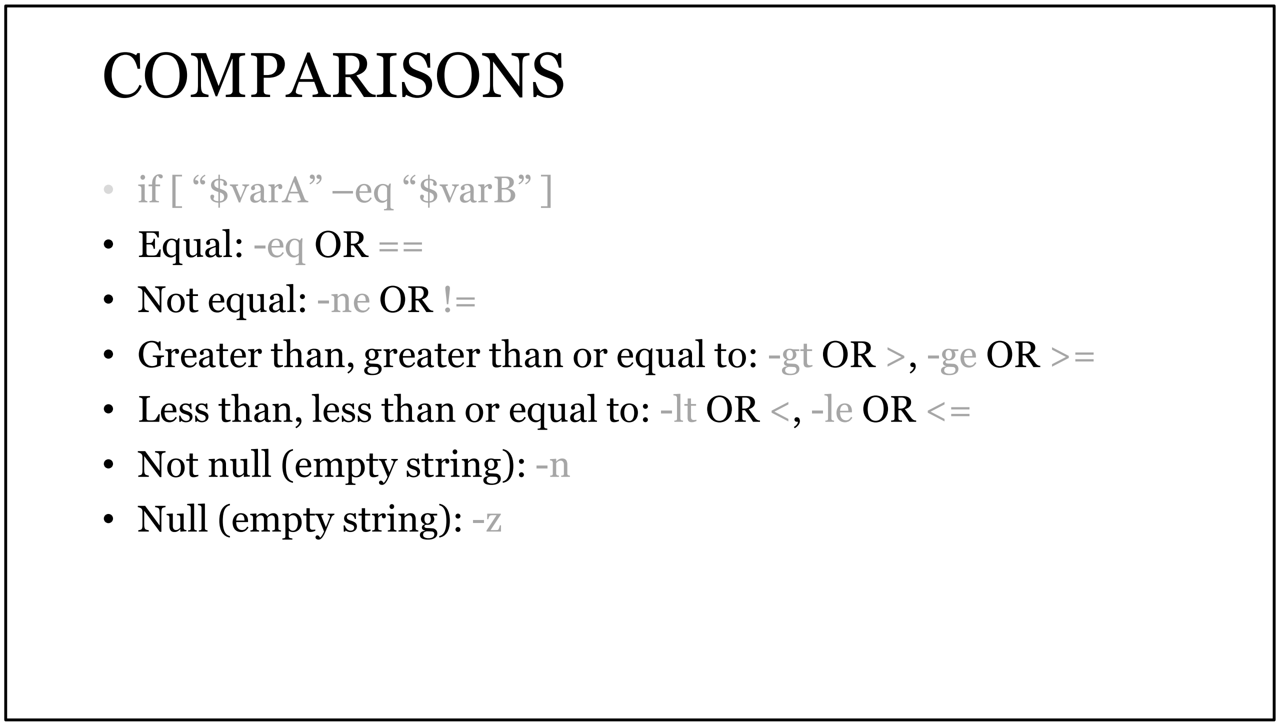
4.4.3 Bash Substitutions



COMMON OPERATIONS

- String operations
  - Concatenate var=“Hello” ; var=“$var World”
  - Length ${#string} OR expr length $string ex. ${#name}
  - Extract a substring echo ${string:position} ex. ${name:3}
  - Replacing substring ${string/substring/replacement} ex. ${name/ch/xx}
- Compound operations
  - AND: -a
  - OR: -o

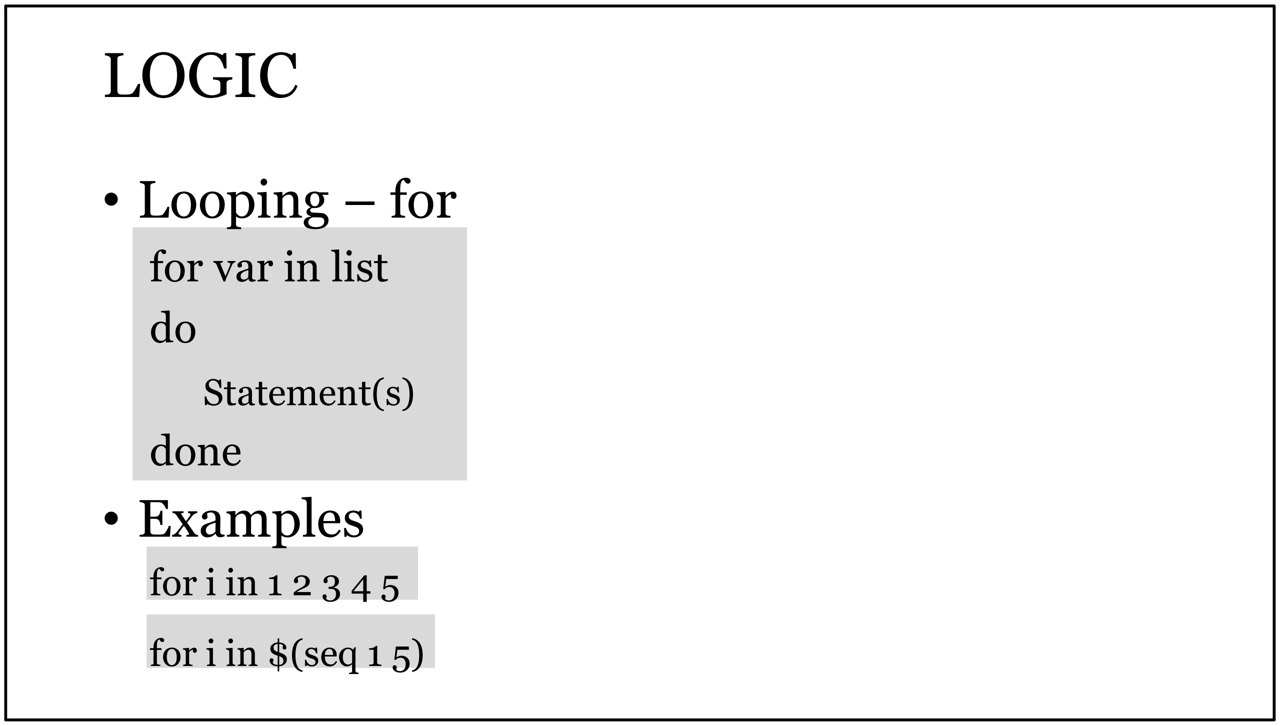
4.4.5 Bash Common operations
4.4.5.1 String operations



COMPARISONS

- if [“$varA” –eq “$varB”]
- Equal: -eq OR ==
- Not equal: -ne OR !=
- Greater than, greater than or equal to: -gt OR >, -ge OR >=
- Less than, less than or equal to: -lt OR <, -le OR <=
- Not null (empty string): -n
- Null (empty string): -z

4.4.5 Bash Common operations
4.4.5.2 Comparisons



LOGIC

- Looping – for

```
for var in list
do
    Statement(s)
done
```

- Examples

```
for i in 1 2 3 4 5
```

```
for i in $(seq 1 5)
```

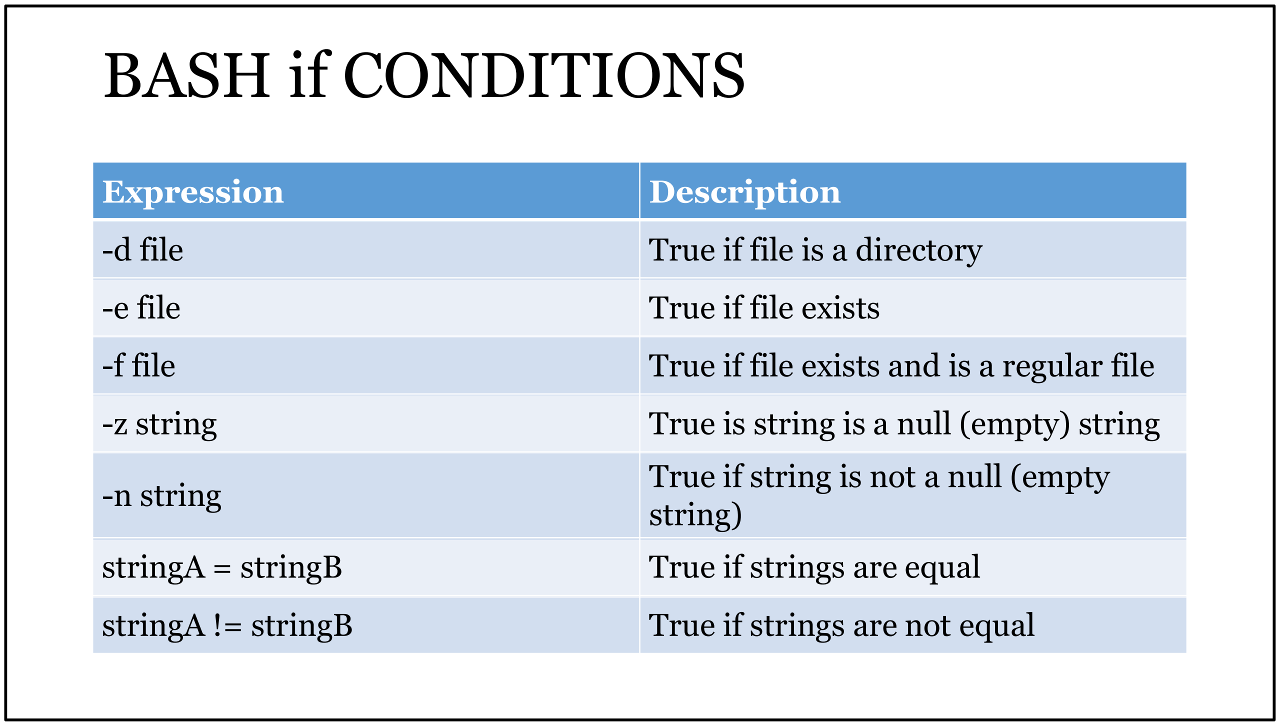
4.4.1 Bash Logic
4.4.1.1 Looping

```
for i in 1 2 3 4 5
for i in {1..5}
for (( ctr=1; ctr<=10; c++ ))
```

FLOW CONTROL

```
if condition
then
    commands
elif commands
then
    commands
else
    commands
fi
```

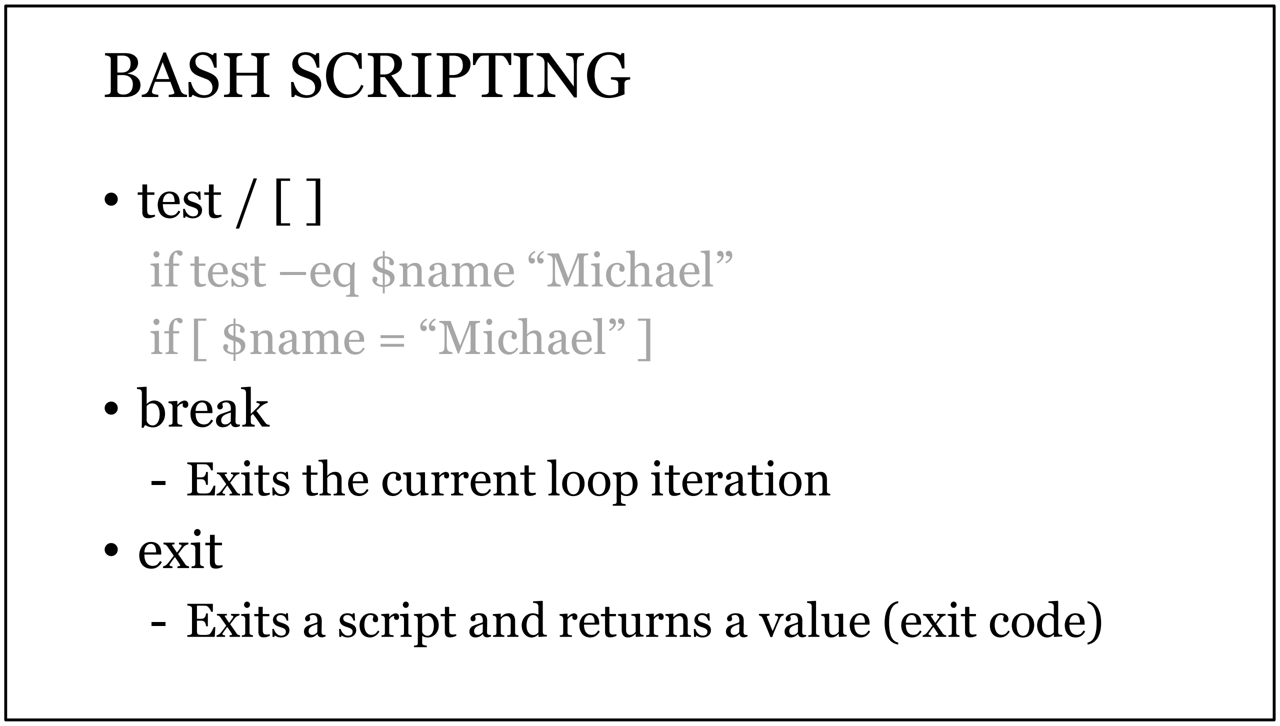
4.4.1 Bash Logic
4.4.1.2 Flow control



BASH if CONDITIONS

Expression	Description
-d file	True if file is a directory
-e file	True if file exists
-f file	True if file exists and is a regular file
-z string	True is string is a null (empty) string
-n string	True if string is not a null (empty string)
stringA = stringB	True if strings are equal
stringA != stringB	True if strings are not equal

4.4.1 Bash Logic
4.4.1.2 Flow control



BASH SCRIPTING

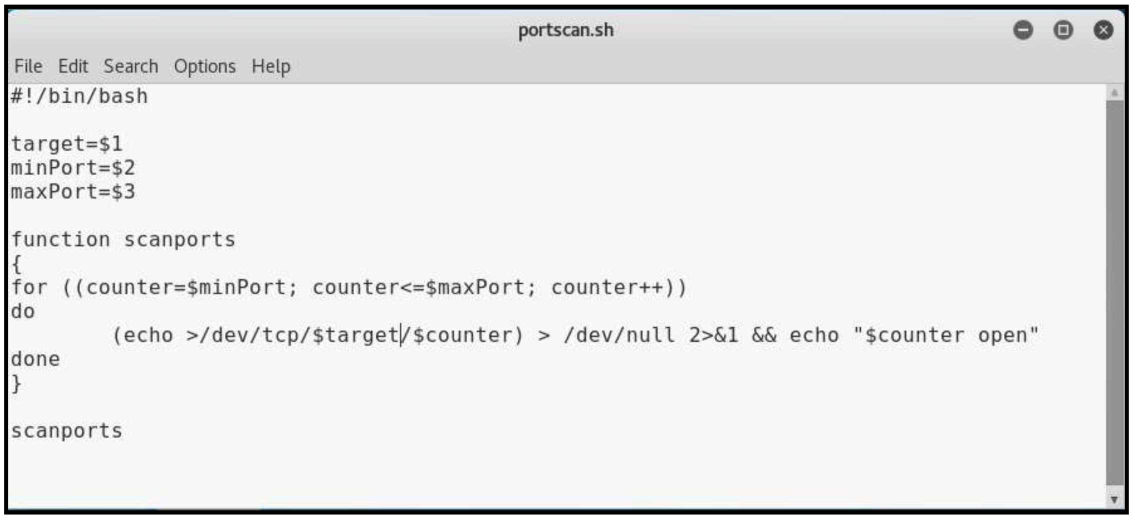
- test / [ ]

  ```
  if test –eq $name "Michael"
  if [ $name = "Michael" ]
  ```

- break
  - Exits the current loop iteration
- exit
  - Exits a script and returns a value (exit code)

4.4.1 Bash Logic
4.4.1.2 Flow control

BASH PORT SCANNER

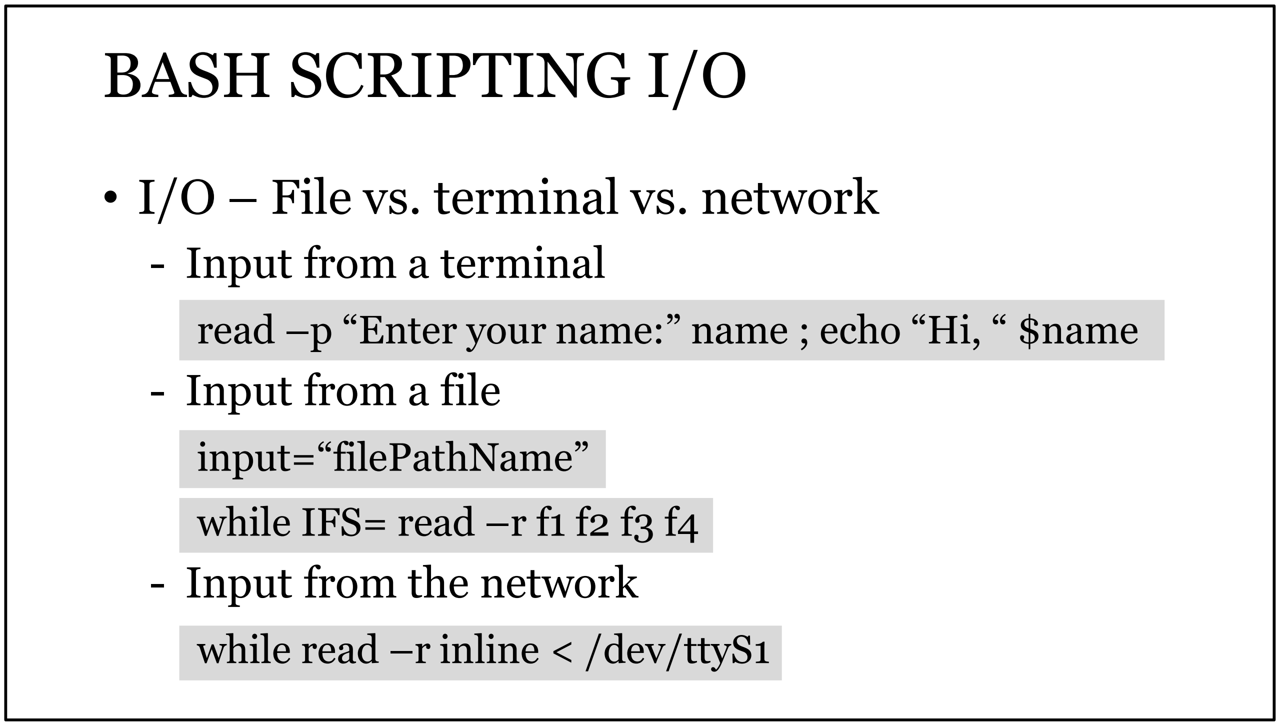


```
#!/bin/bash

target=$1
minPort=$2
maxPort=$3

function scanports
{
for ((counter=$minPort; counter<=$maxPort; counter++))
do
        (echo >/dev/tcp/$target/$counter) > /dev/null 2>&1 && echo "$counter open"
done
}

scanports
```

https://pentestlab.blog/2012/11/12/creating-a-tcp-port-scanner-in-bash/

Bash Scripting Techniques

Episode 3



BASH SCRIPTING I/O

- I/O – File vs. terminal vs. network
  - Input from a terminal

```
read –p “Enter your name:” name ; echo “Hi, “ $name
```

  - Input from a file

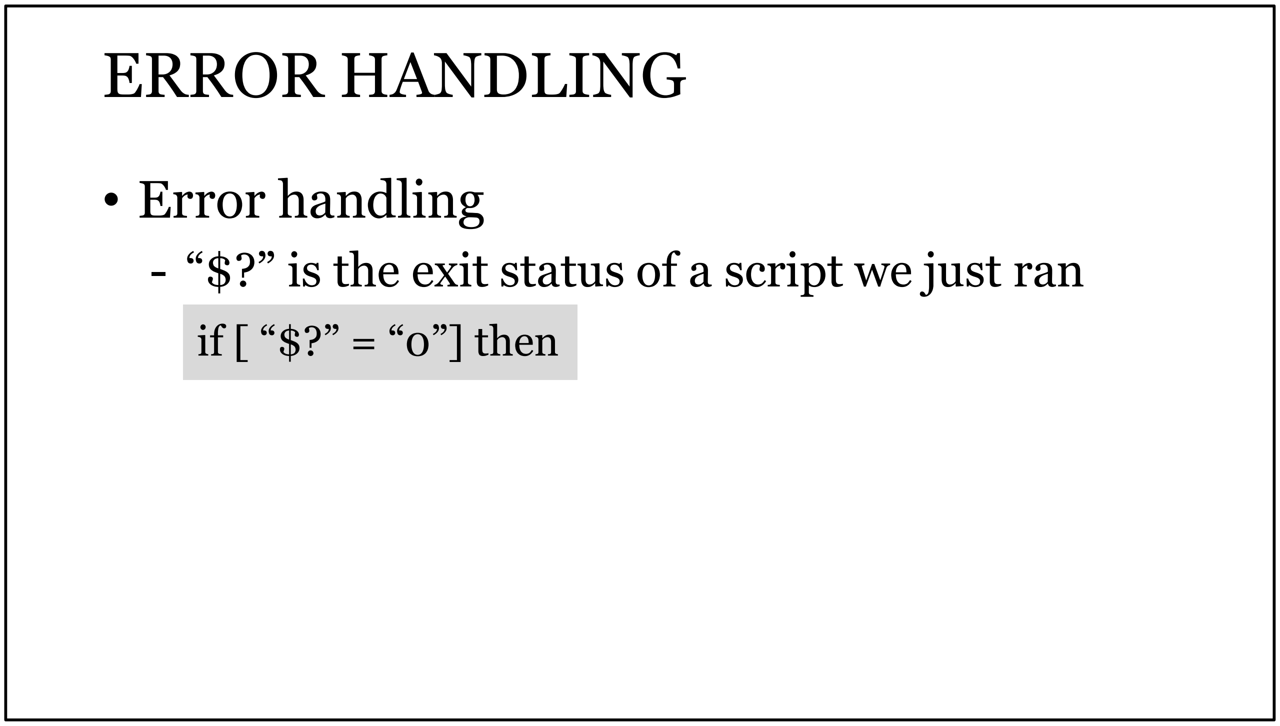
```
input=“filePathName”
```

```
while IFS= read –r f1 f2 f3 f4
```

  - Input from the network

```
while read –r inline < /dev/ttyS1
```

4.4.2 Bash I/O
4.4.2.1 File vs. terminal vs. network



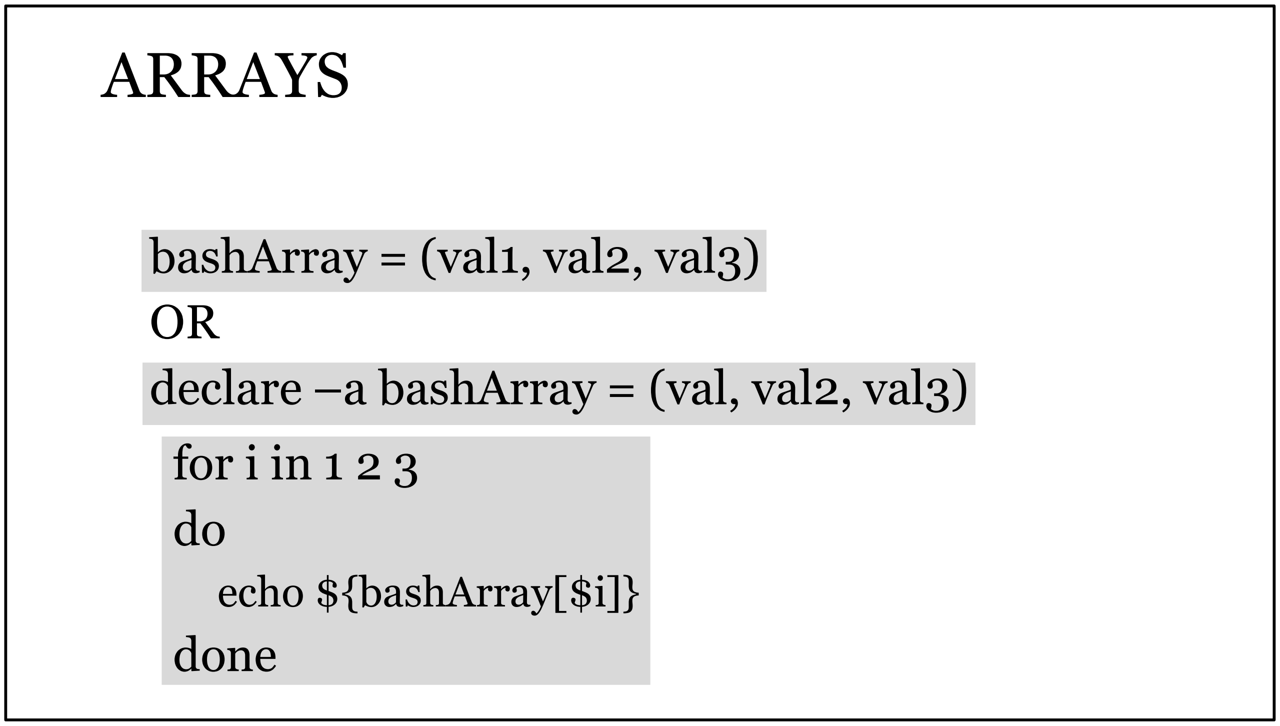
ERROR HANDLING

- Error handling
  - “$?” is the exit status of a script we just ran

```
if [ “$?” = “0”] then
```

4.4.6 Bash Error handling
4.4.7 Bash Arrays

```
bashArray = (val1, val2, val3)             OR declare –a bashArray = (val, val2, val3)
arrayLength=${#array[@]}
for i in ${bashArray[@]}
do
        echo $i
done
```



ARRAYS

```
bashArray = (val1, val2, val3)
OR
declare –a bashArray = (val, val2, val3)
for i in 1 2 3
do
   echo ${bashArray[$i]}
done
```

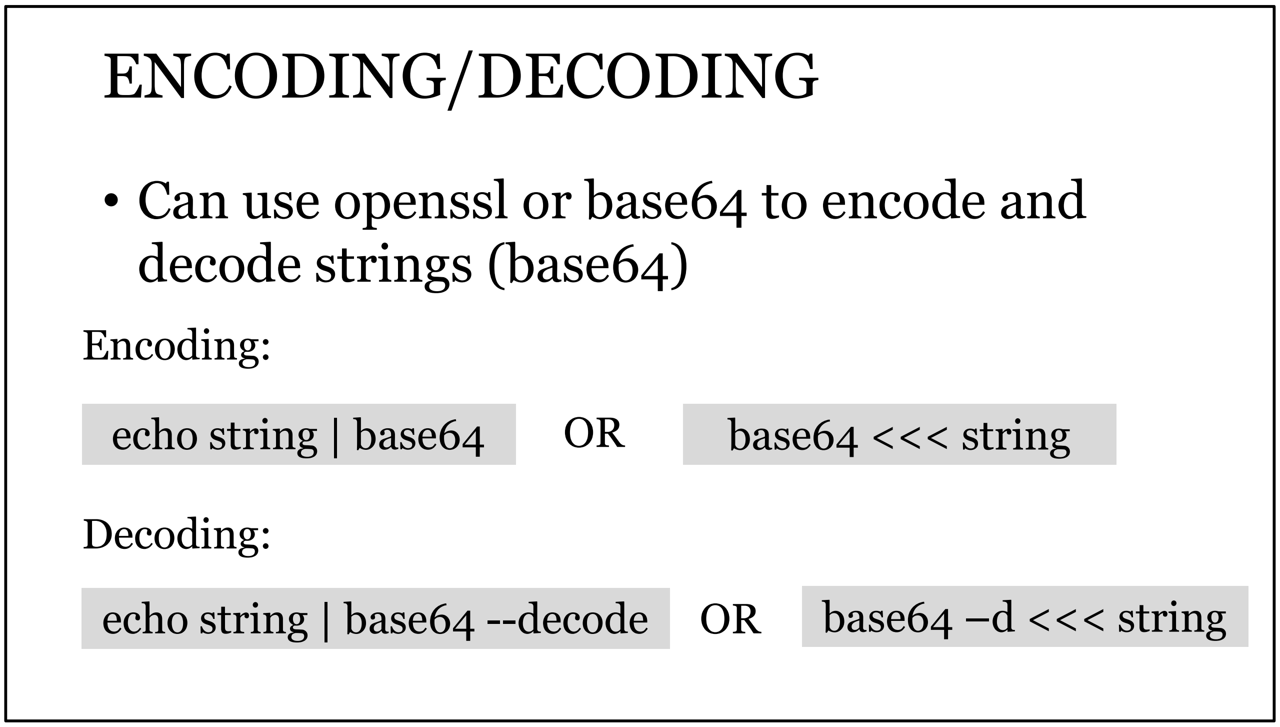
4.4.6 Bash Error handling
4.4.7 Bash Arrays

```
bashArray = (val1, val2, val3)            OR declare –a bashArray = (val, val2, val3)
arrayLength=${#array[@]}
for i in ${bashArray[@]}
do
        echo $i
done
```

ENCODING/DECODING

- locale – shows local related environment variables
- Can change assignment of LANG for local character encoding
  - Allows bash to accept special charaters (i.e. LANG=da_DK.UTF-8)

4.4.8 Bash Encoding/decoding



ENCODING/DECODING

- Can use openssl or base64 to encode and decode strings (base64)

Encoding:

```
echo string | base64
```

OR

```
base64 <<< string
```

Decoding:

```
echo string | base64 --decode
```

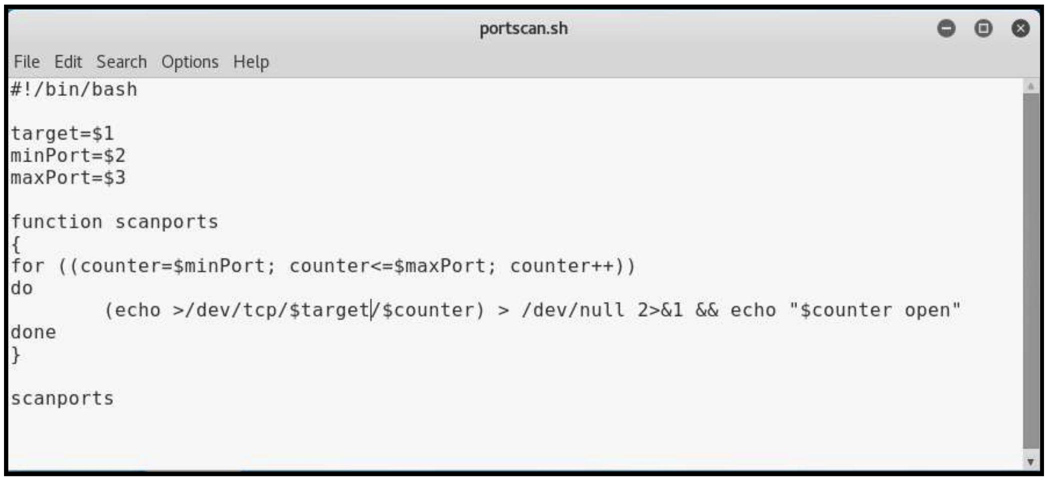
OR

```
base64 –d <<< string
```

4.4.8 Bash Encoding/decoding

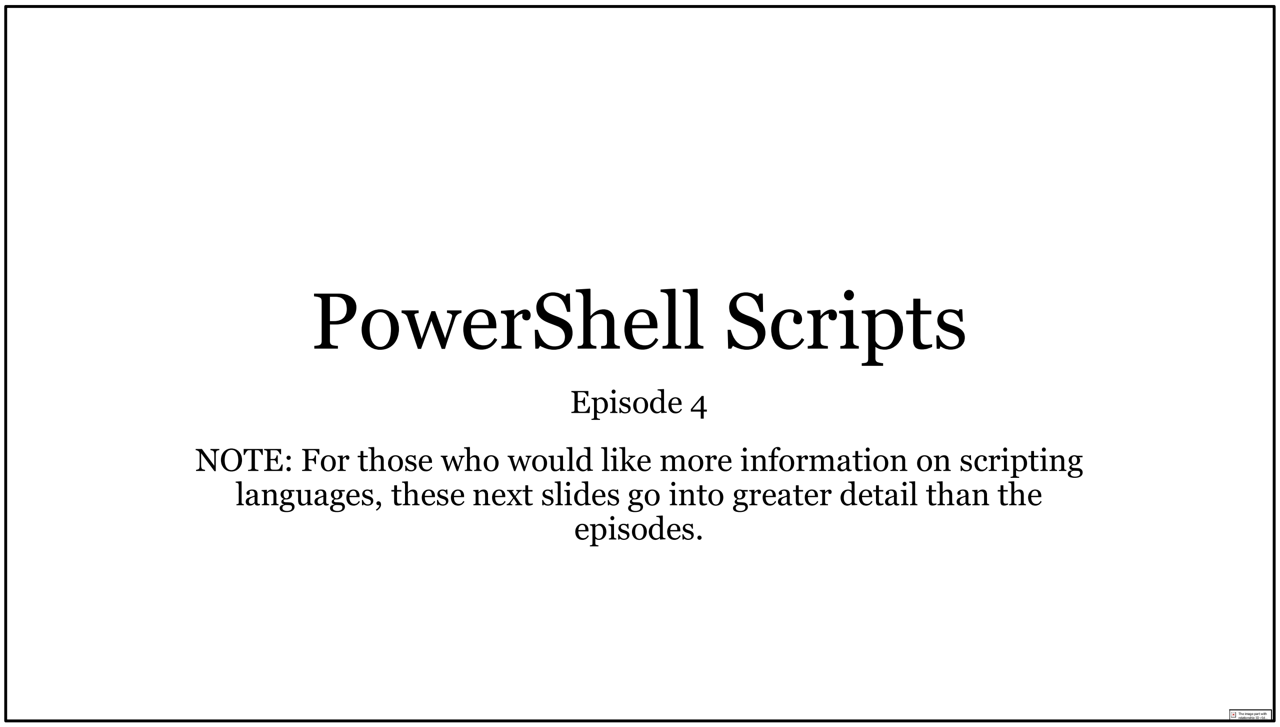
BASH: PUTTING IT ALL TOGETHER

- Port scanner in bash



```
#!/bin/bash

target=$1
minPort=$2
maxPort=$3

function scanports
{
for ((counter=$minPort; counter<=$maxPort; counter++))
do
        (echo >/dev/tcp/$target/$counter) > /dev/null 2>&1 && echo "$counter open"
done
}

scanports
```

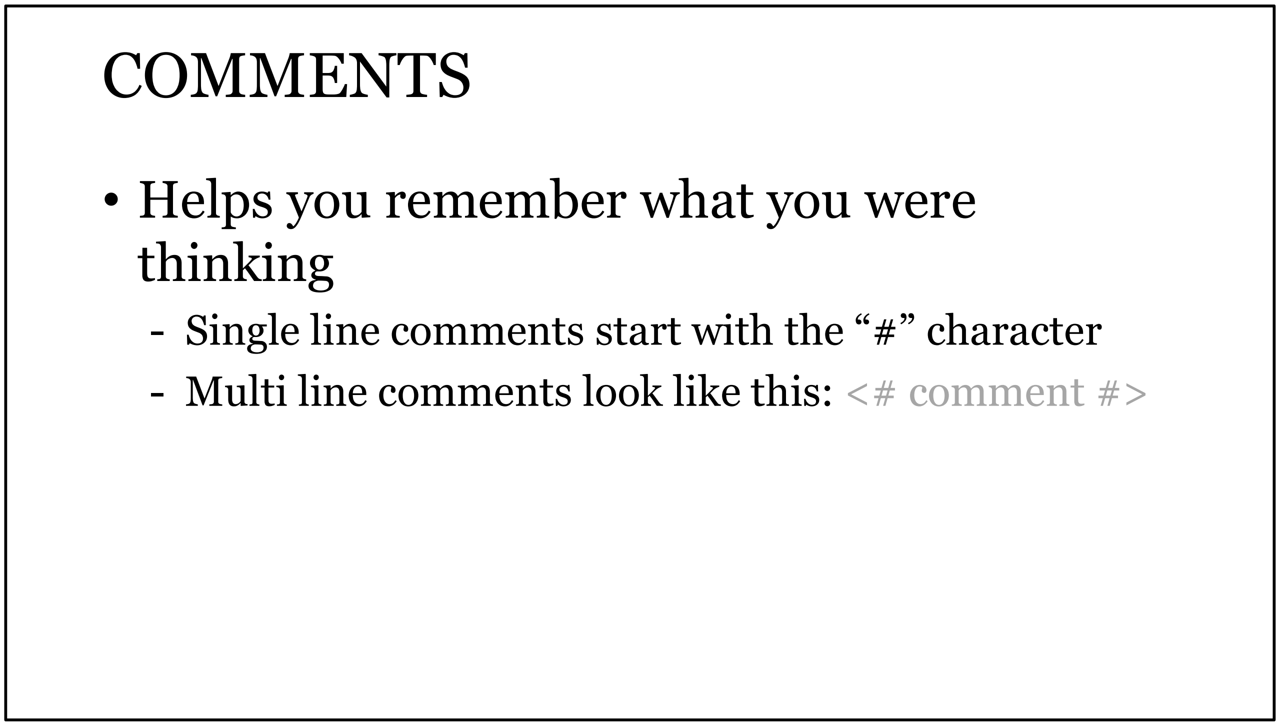
https://pentestlab.blog/2012/11/12/creating-a-tcp-port-scanner-in-bash/



PowerShell Scripts

Episode 4

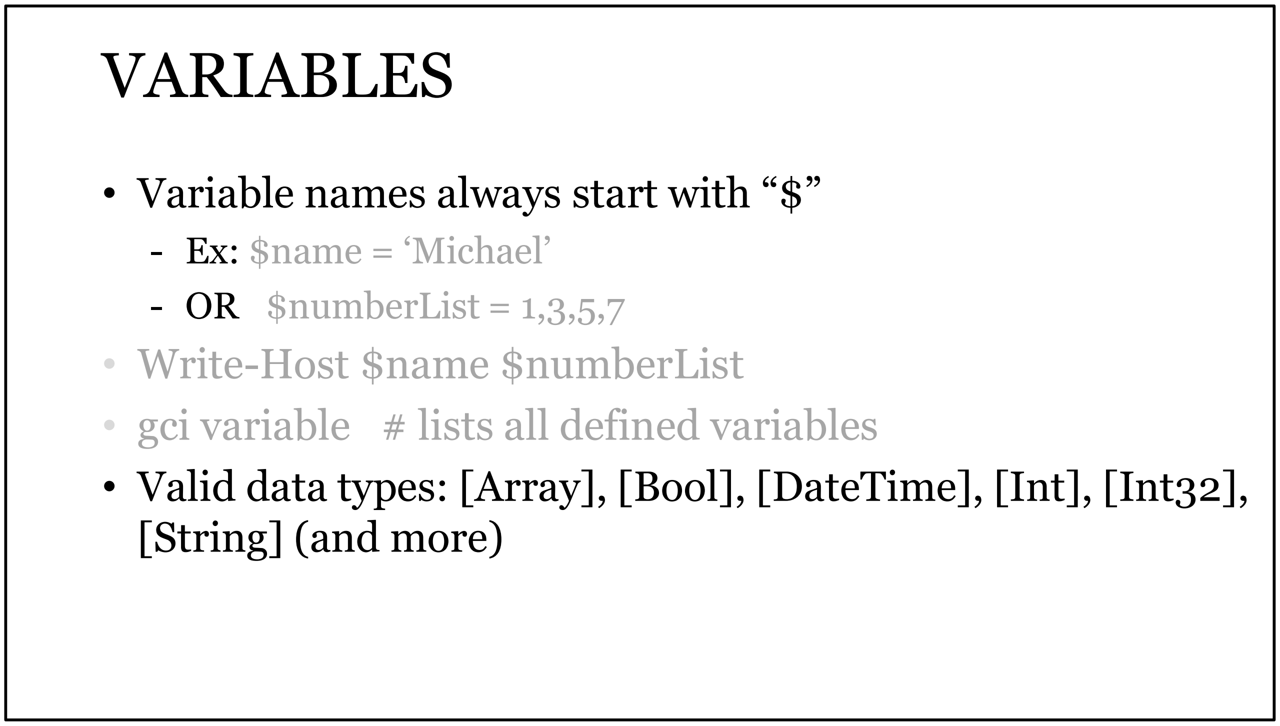
NOTE: For those who would like more information on scripting languages, these next slides go into greater detail than the episodes.



COMMENTS

- Helps you remember what you were thinking
  - Single line comments start with the “#” character
  - Multi line comments look like this: <# comment #>

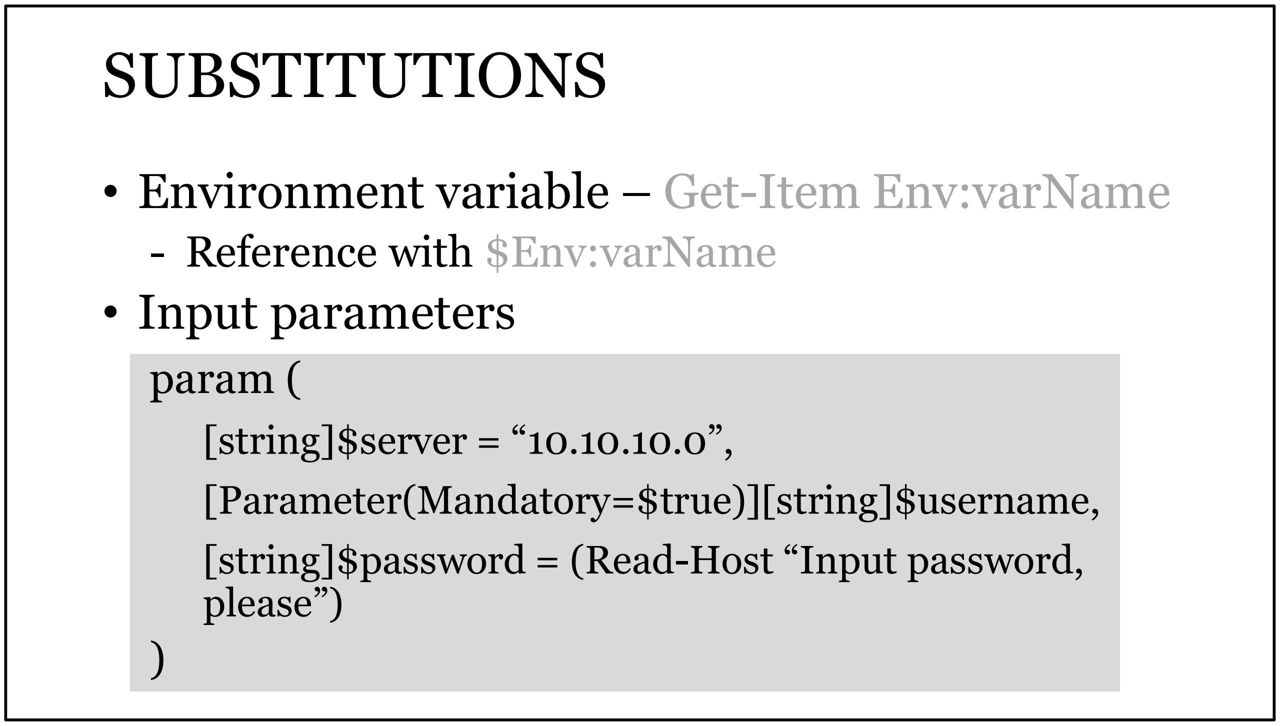
4.4.4 PowerShell Variables



VARIABLES

- Variable names always start with “$”
  - Ex: $name = ‘Michael’
  - OR $numberList = 1,3,5,7
- Write-Host $name $numberList
- gci variable # lists all defined variables
- Valid data types: [Array], [Bool], [DateTime], [Int], [Int32], [String] (and more)

4.4.4 PowerShell Variables

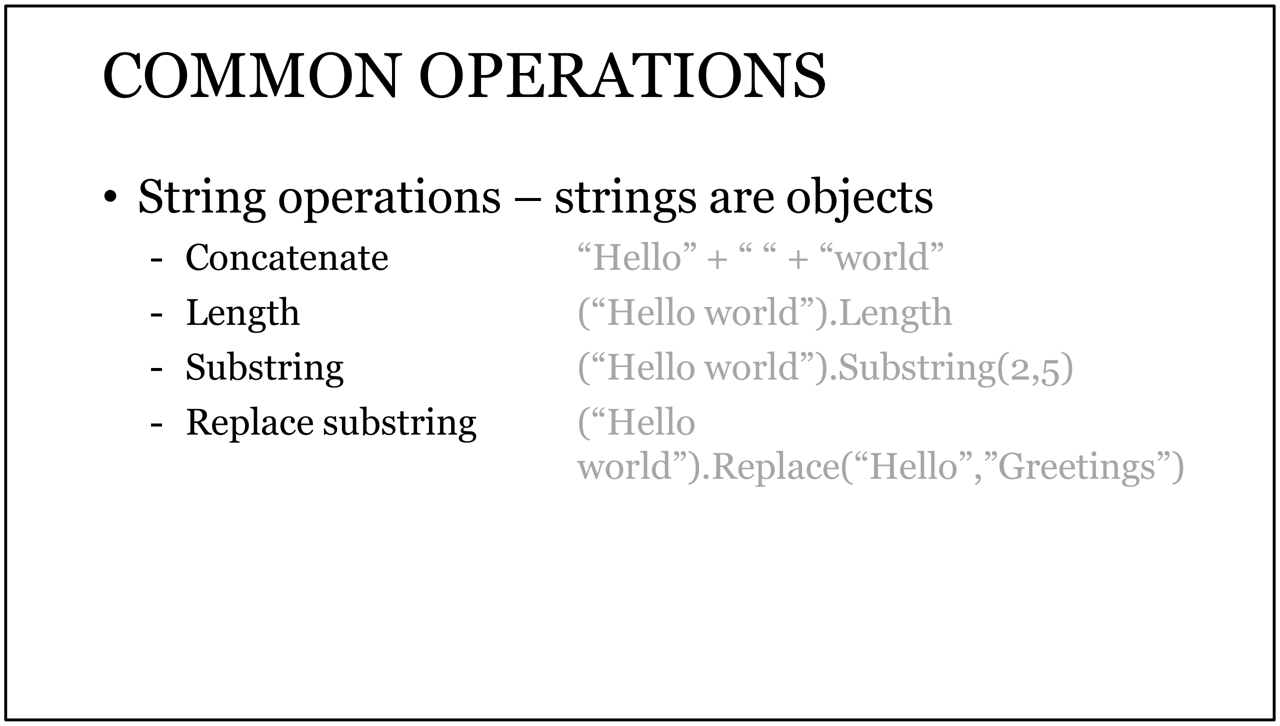


SUBSTITUTIONS

- Environment variable – Get-Item Env:varName
  - Reference with $Env:varName
- Input parameters

```
param (
    [string]$server = “10.10.10.0”,
    [Parameter(Mandatory=$true)][string]$username,
    [string]$password = (Read-Host “Input password,
    please”)
)
```

4.4.3 PowerShell Substitutions

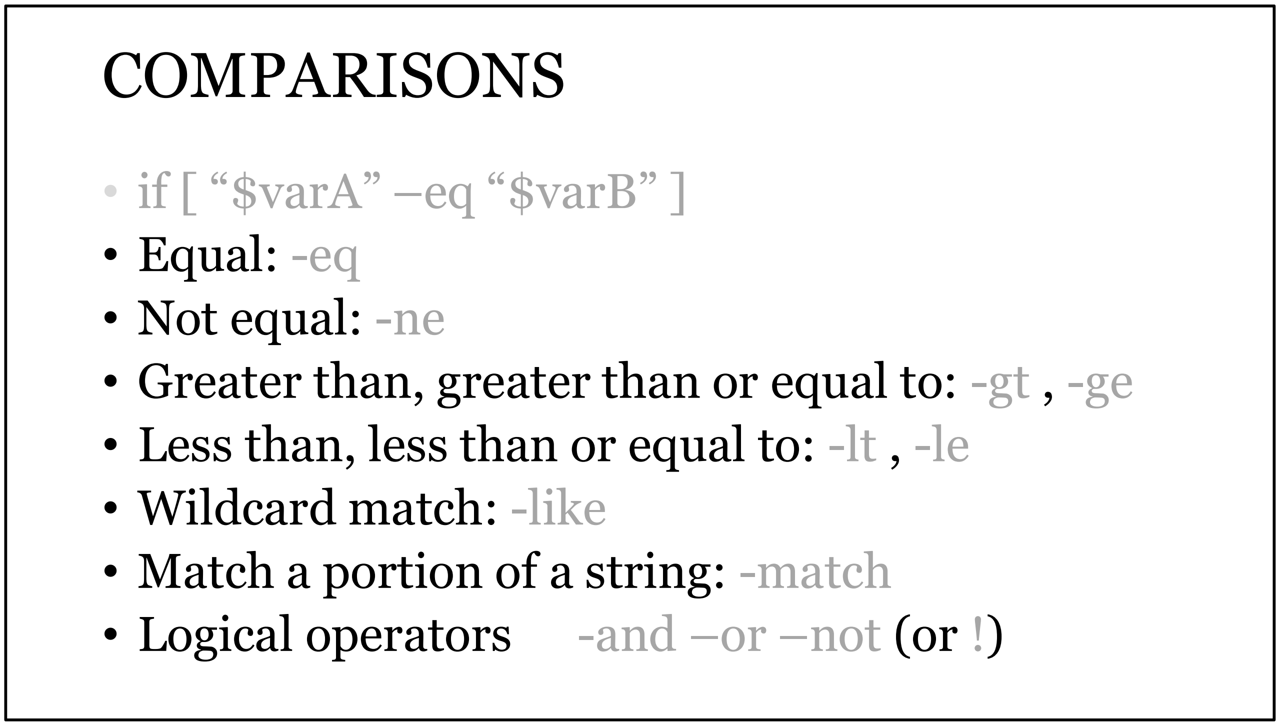


COMMON OPERATIONS

- String operations – strings are objects

- Concatenate	“Hello” + “ “ + “world”
- Length	(“Hello world”).Length
- Substring	(“Hello world”).Substring(2,5)
- Replace substring	(“Hello world”).Replace(“Hello”,”Greetings”)

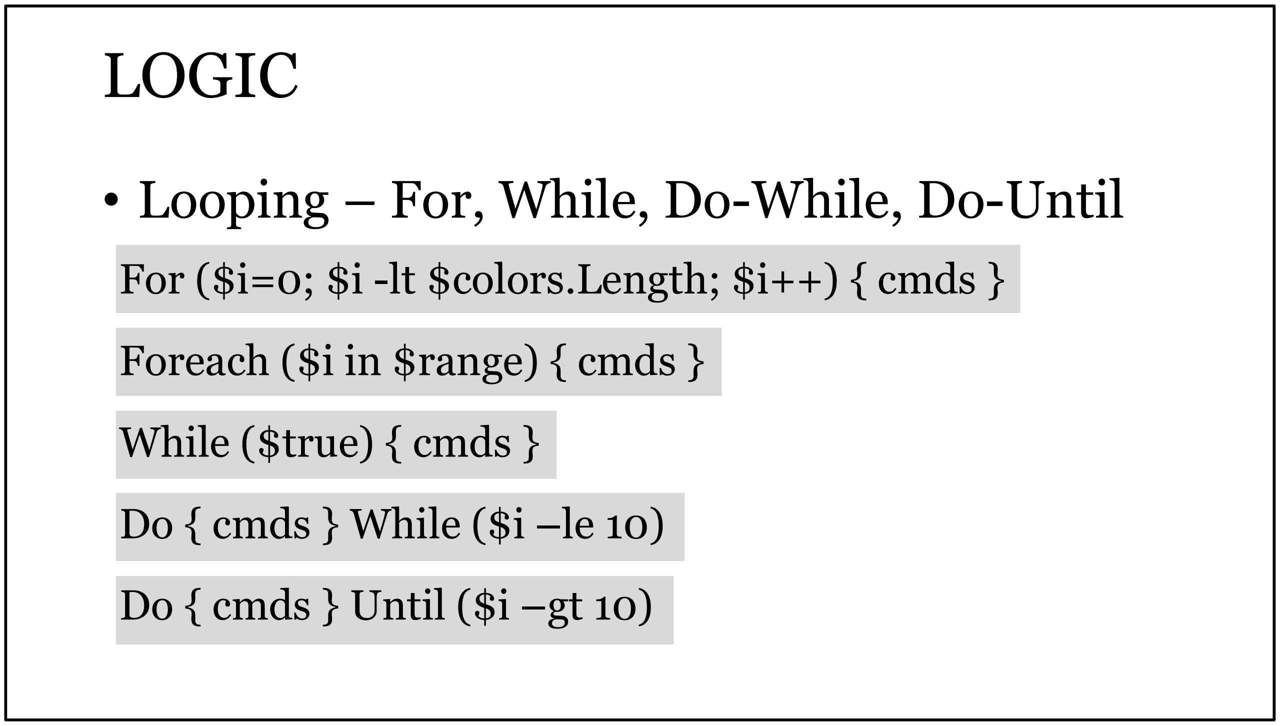
4.4.5 PowerShell Common operations
4.4.5.1 String operations



COMPARISONS

- if [“$varA” –eq “$varB”]
- Equal: -eq
- Not equal: -ne
- Greater than, greater than or equal to: -gt , -ge
- Less than, less than or equal to: -lt , -le
- Wildcard match: -like
- Match a portion of a string: -match
- Logical operators -and –or –not (or !)

4.4.5 PowerShell Common operations
4.4.5.2 Comparisons



LOGIC

- Looping – For, While, Do-While, Do-Until

```
For ($i=0; $i -lt $colors.Length; $i++) { cmds }
Foreach ($i in $range) { cmds }
While ($true) { cmds }
Do { cmds } While ($i –le 10)
Do { cmds } Until ($i –gt 10)
```

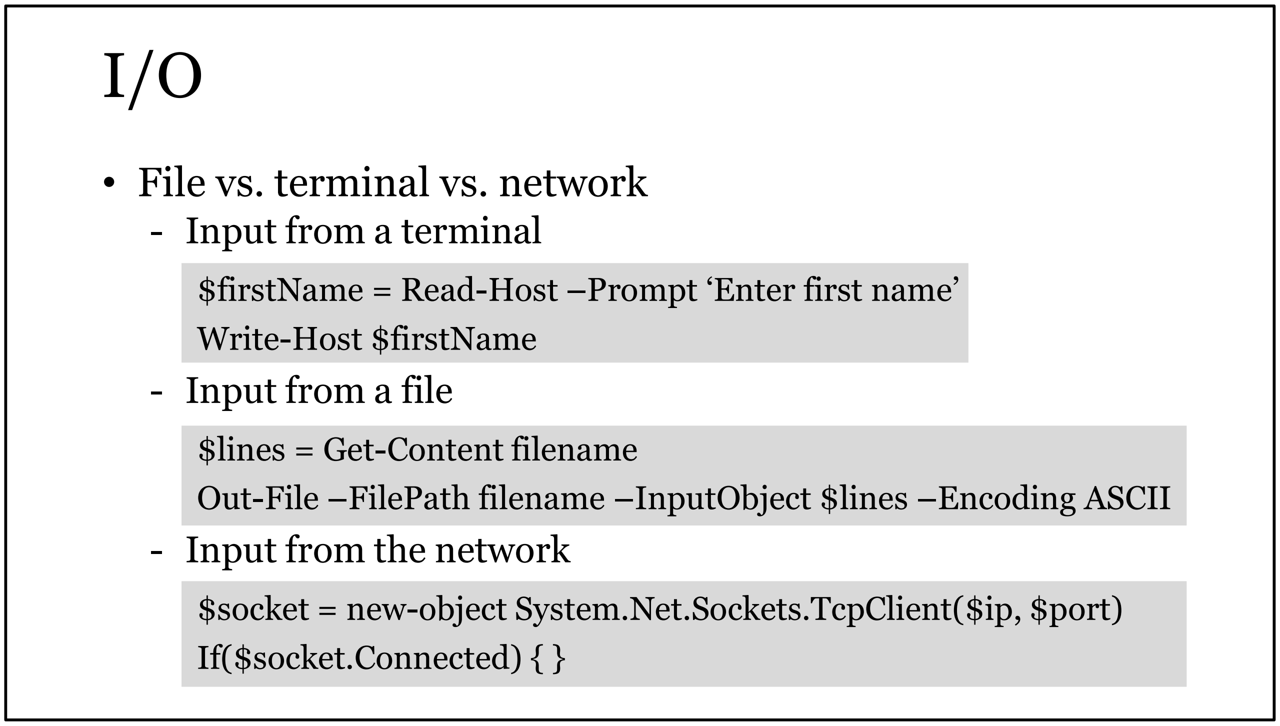
4.4.1 PowerShell Logic
4.4.1.1 Looping

LOGIC

- Flow control

```
if (condition) {
    statements
} elseif (condition) {
    statements
} else {
    statements
}
```

4.4.1 PowerShell Logic
4.4.1.2 Flow control



I/O

- File vs. terminal vs. network
  - Input from a terminal

```
$firstName = Read-Host –Prompt ‘Enter first name’
Write-Host $firstName
```

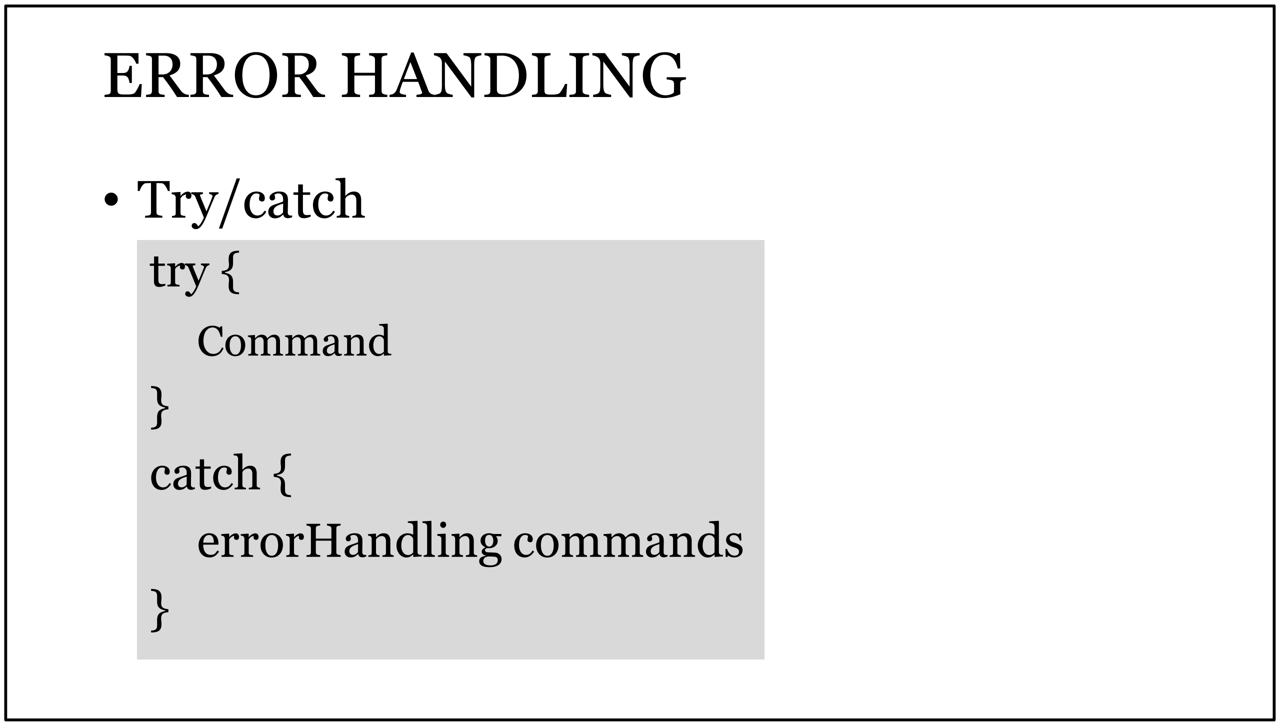
  - Input from a file

```
$lines = Get-Content filename
Out-File –FilePath filename –InputObject $lines –Encoding ASCII
```

  - Input from the network

```
$socket = new-object System.Net.Sockets.TcpClient($ip, $port)
If($socket.Connected) { }
```

4.4.2 PowerShell I/O
4.4.2.1 File vs. terminal vs. network



ERROR HANDLING

- Try/catch

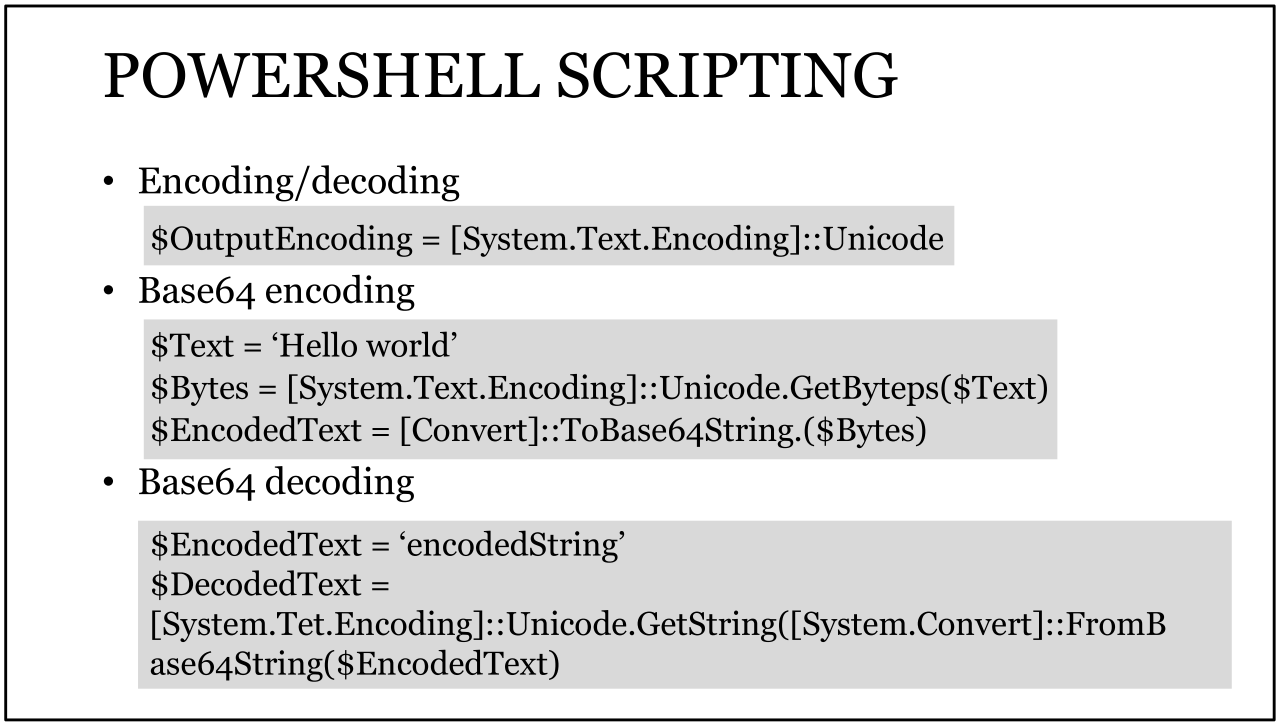
```
try {
    Command
}
catch {
    errorHandling commands
}
```

4.4.6 PowerShell Error handling

ARRAYS

```
$PSarray=@(1.3.5.7.9);
$PSarray.Length
for ($i = 0; $i –lt $PSarray.Length; $i++) {
   $PSarray[$i]
}
foreach ($element in $PSarray) {
       $element
}
```

4.4.7 PowerShell Arrays



POWERSHELL SCRIPTING

- Encoding/decoding

```
$OutputEncoding = [System.Text.Encoding]::Unicode
```

- Base64 encoding

```
$Text = ‘Hello world’
$Bytes = [System.Text.Encoding]::Unicode.GetByteps($Text)
$EncodedText = [Convert]::ToBase64String.($Bytes)
```

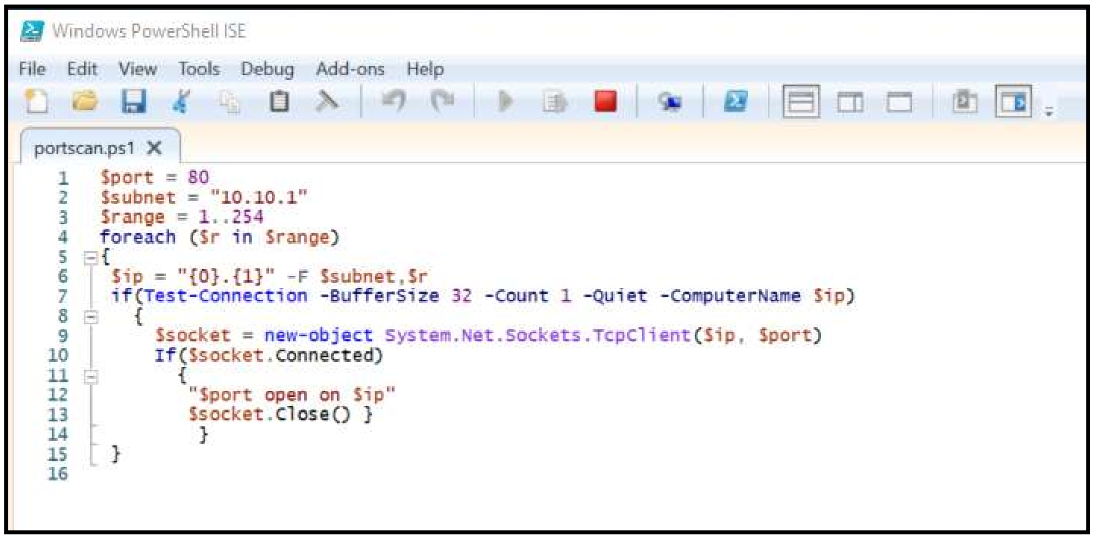
- Base64 decoding

```
$EncodedText = ‘encodedString’
$DecodedText =
[System.Tet.Encoding]::Unicode.GetString([System.Convert]::FromB
ase64String($EncodedText)
```

4.4.8 PowerShell Encoding/decoding

https://adsecurity.org/?p=478

PowerShell: Putting it all together

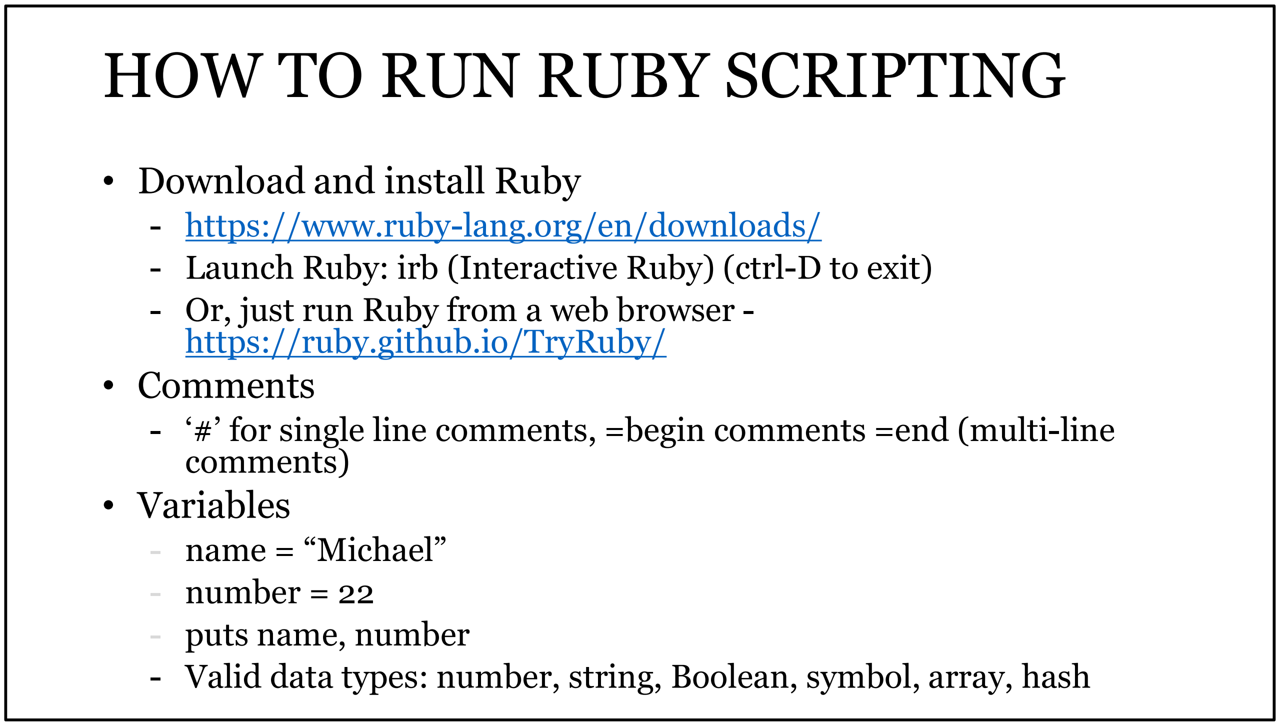


```
$port = 80
$subnet = "10.10.1"
$range = 1..254
foreach ($r in $range)
{
 $ip = "{0}.{1}" -F $subnet,$r
 if(Test-Connection -BufferSize 32 -Count 1 -Quiet -ComputerName $ip)
   {
     $socket = new-object System.Net.Sockets.TcpClient($ip, $port)
     If($socket.Connected)
       {
         "$port open on $ip"
         $socket.Close() }
          }
 }
```

```
# -----------------------------------------------------------------------------
# Script: PoshPortScanner.ps1
# Author: ed wilson, msft
# Date: 02/19/2014 15:17:33
# Keywords: Security, Networking, Tcp/IP, Monitoring
# comments: This script scans a range of IP addresses for web servers listening
# to port 80. It is a useful audit tool, because there are lots of software and
# devices that setup web servers for management, but that do not necessarily
# inform about them.
#
# -----------------------------------------------------------------------------
$port = 80
$net = "192.168.0"
$range = 1..254
foreach ($r in $range)
{
 $ip = "{0}.{1}" -F $net,$r
 if(Test-Connection -BufferSize 32 -Count 1 -Quiet -ComputerName $ip)
   {
    $socket = new-object System.Net.Sockets.TcpClient($ip, $port)
    If($socket.Connected)
      {
       "$ip listening to port $port"
       $socket.Close() }
        }
}
```

Ruby Scripts

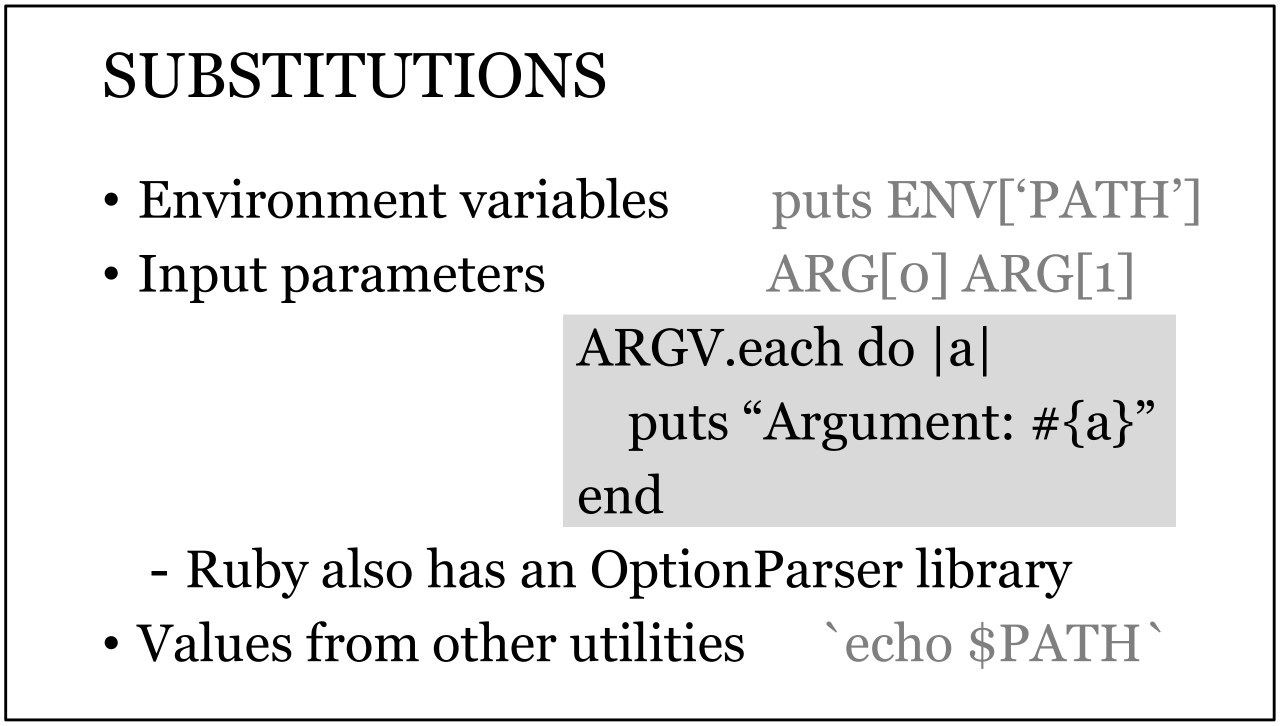
Episode 5



HOW TO RUN RUBY SCRIPTING

- Download and install Ruby
  - https://www.ruby-lang.org/en/downloads/
  - Launch Ruby: irb (Interactive Ruby) (ctrl-D to exit)
  - Or, just run Ruby from a web browser - https://ruby.github.io/TryRuby/
- Comments
  - '#' for single line comments, =begin comments =end (multi-line comments)
- Variables
  - name = "Michael"
  - number = 22
  - puts name, number
  - Valid data types: number, string, Boolean, symbol, array, hash

4.4.4 Ruby Variables



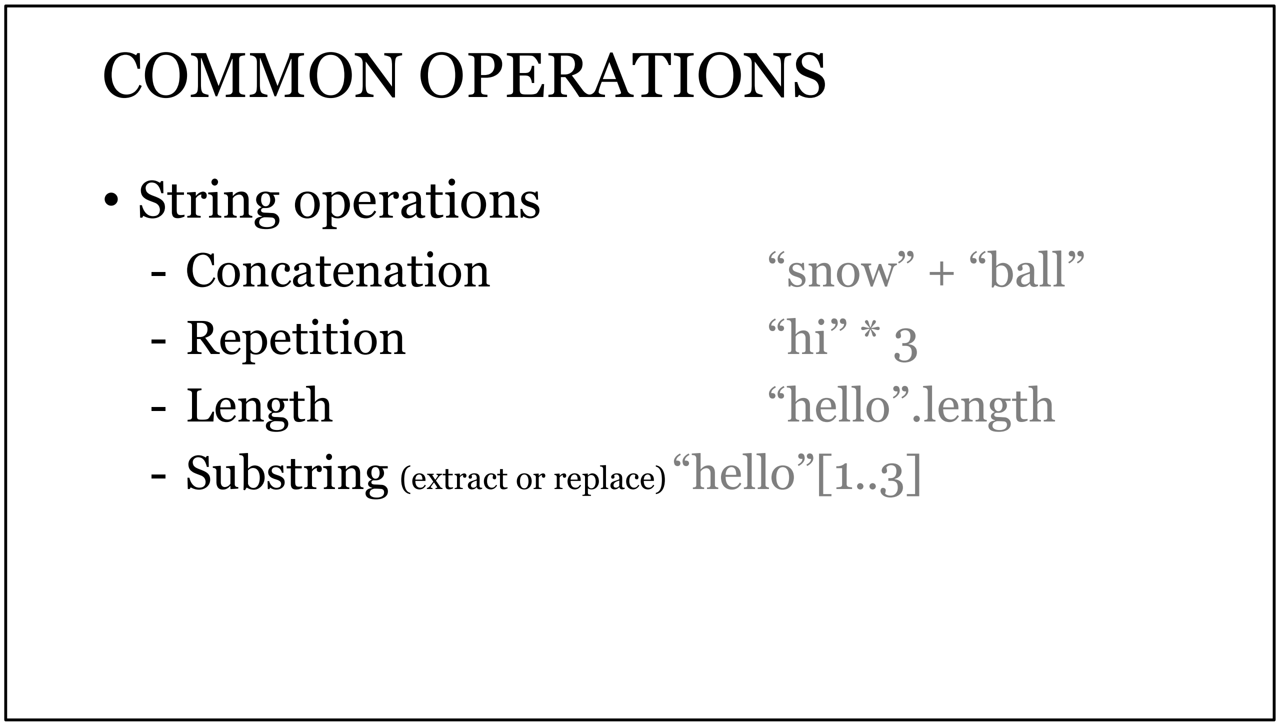
SUBSTITUTIONS

- Environment variables puts ENV[‘PATH’]
- Input parameters ARG[0] ARG[1]

```
ARGV.each do |a|
   puts “Argument: #{a}”
end
```

  - Ruby also has an OptionParser library
- Values from other utilities `echo $PATH`

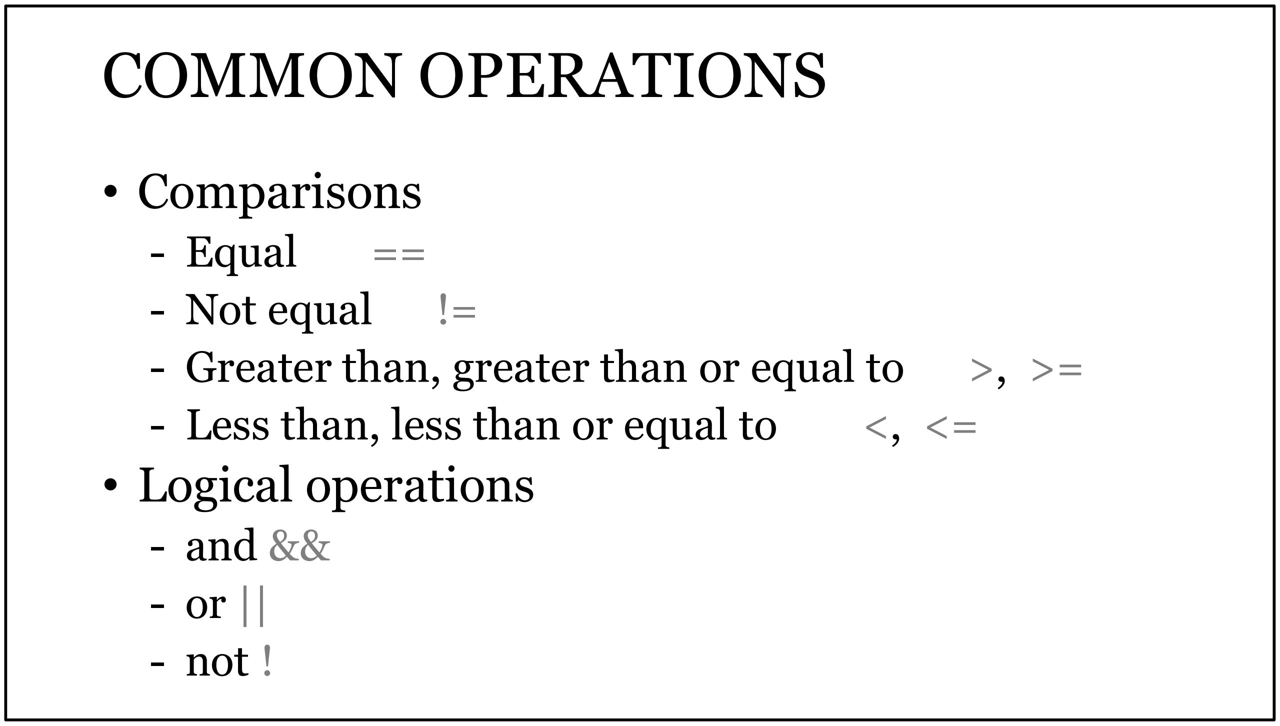
4.4.3 Ruby Substitutions



COMMON OPERATIONS

- String operations
  - Concatenation “snow” + “ball”
  - Repetition “hi” * 3
  - Length “hello”.length
  - Substring (extract or replace) “hello”[1..3]

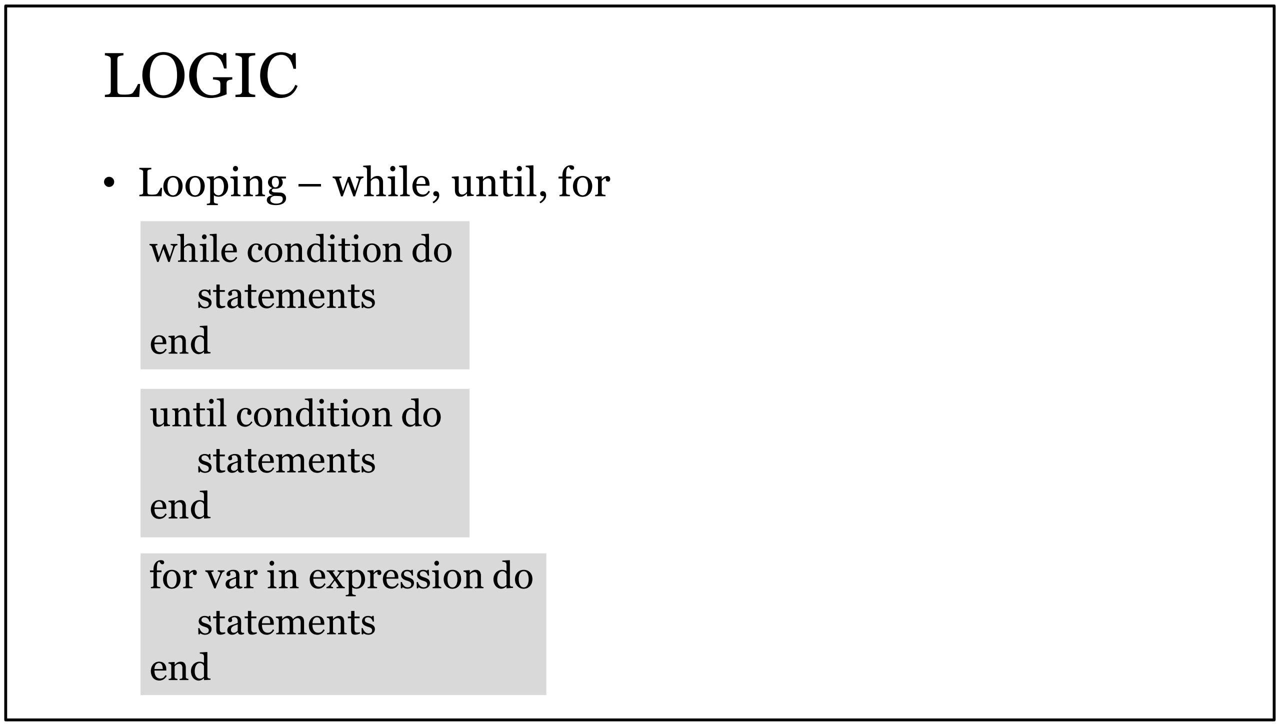
4.4.5 Ruby Common operations
4.4.5.1 String operations



COMMON OPERATIONS

- Comparisons
  - Equal ==
  - Not equal !=
  - Greater than, greater than or equal to >, >=
  - Less than, less than or equal to <, <=
- Logical operations
  - and &&
  - or ||
  - not !

4.4.5 Ruby Common operations
4.4.5.2 Comparisons



LOGIC

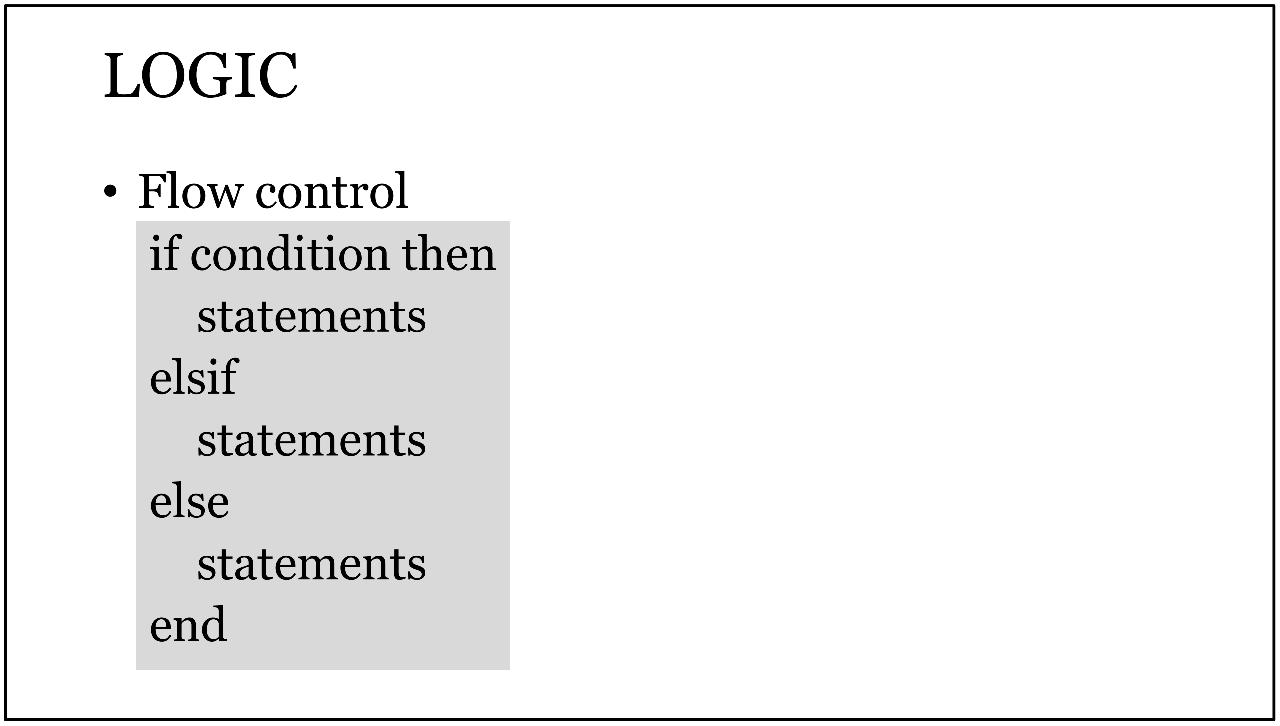
- Looping – while, until, for

```
while condition do
    statements
end
```

```
until condition do
    statements
end
```

```
for var in expression do
    statements
end
```

4.4.1 Ruby Logic
4.4.1.1 Looping



LOGIC

- Flow control

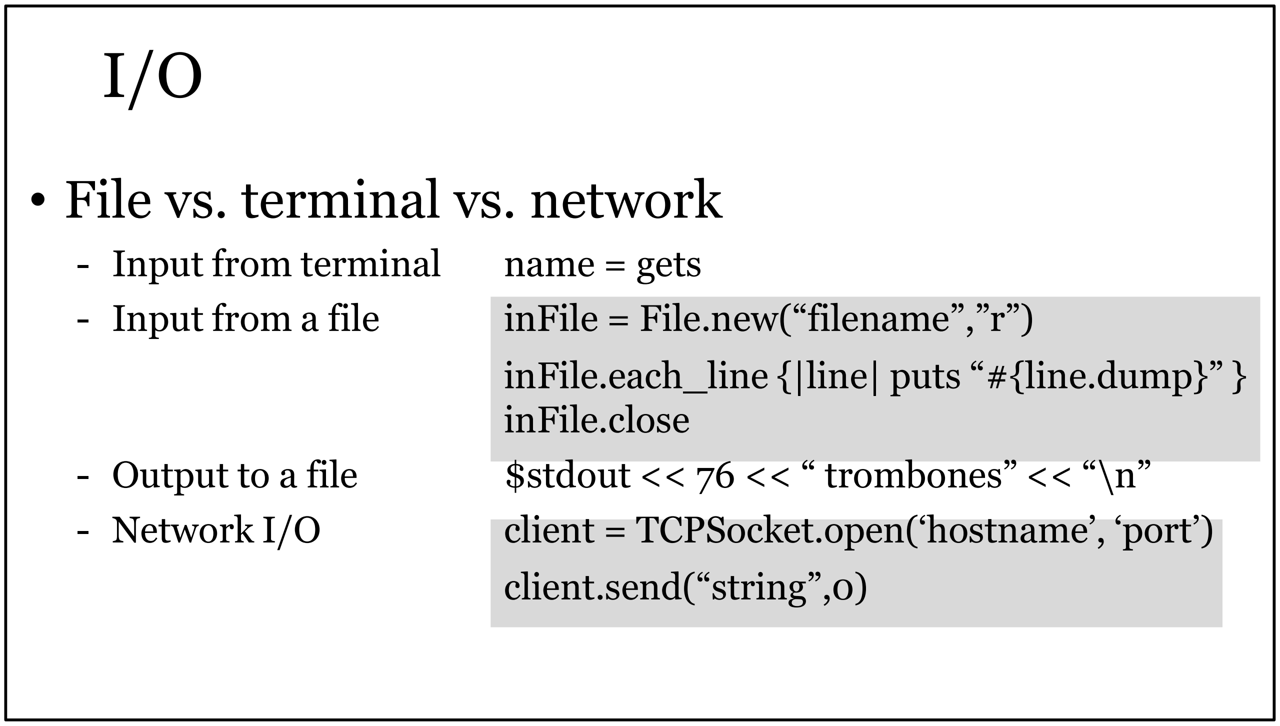
```
if condition then
   statements
elsif
   statements
else
   statements
end
```

4.4.1 Ruby Logic
4.4.1.2 Flow control

LOGIC

```
Case input
   when “A”
         statement
   when “B”
         statement
   else
         statement
end
```

4.4.1 Ruby Logic
4.4.1.2 Flow control



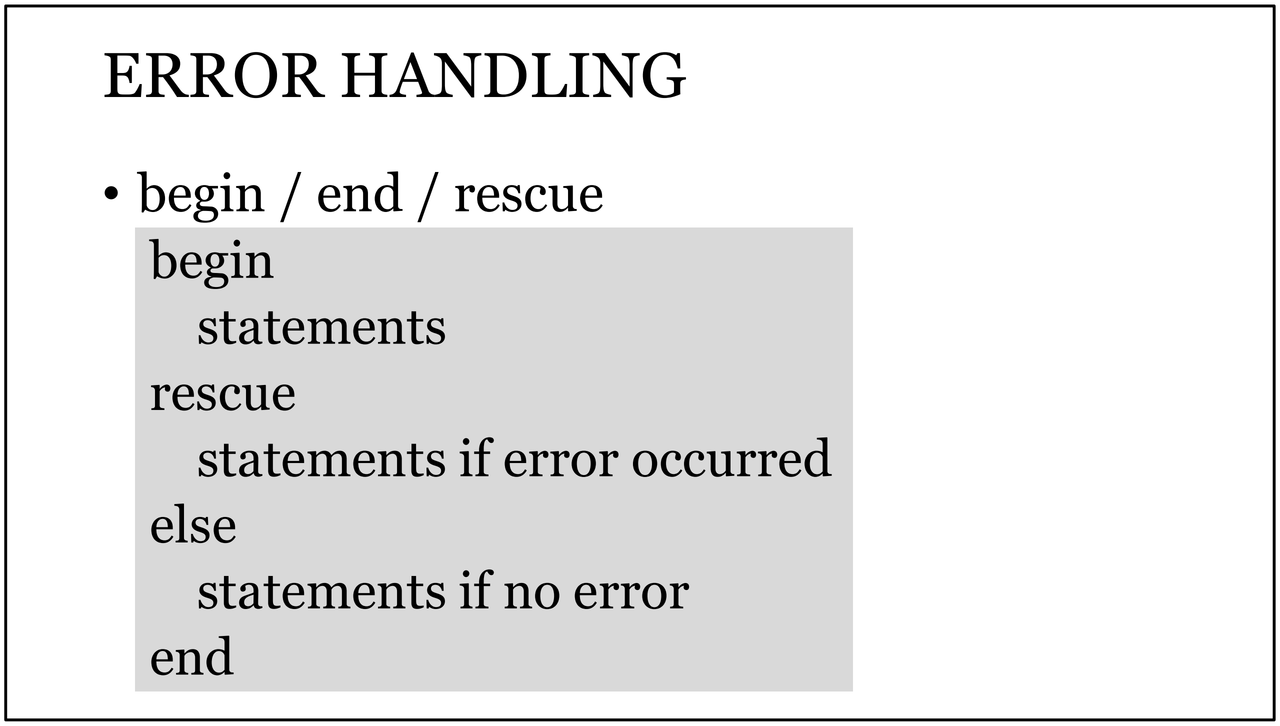
I/O

- File vs. terminal vs. network

- Input from terminal	name = gets
- Input from a file	inFile = File.new(“filename”,”r”) inFile.each_line {|line| puts “#{line.dump}” } inFile.close
- Output to a file	$stdout << 76 << “ trombones” << “\n”
- Network I/O	client = TCPSocket.open(‘hostname’, ‘port’) client.send(“string”,0)

4.4.2 Ruby I/O
4.4.2.1 File vs. terminal vs. network

require 'socket' client = TCPSocket.open('localhost', 'finger')
client.send("oracle\n", 0) # 0 means standard packet puts client.readlines client.close



ERROR HANDLING

- begin / end / rescue

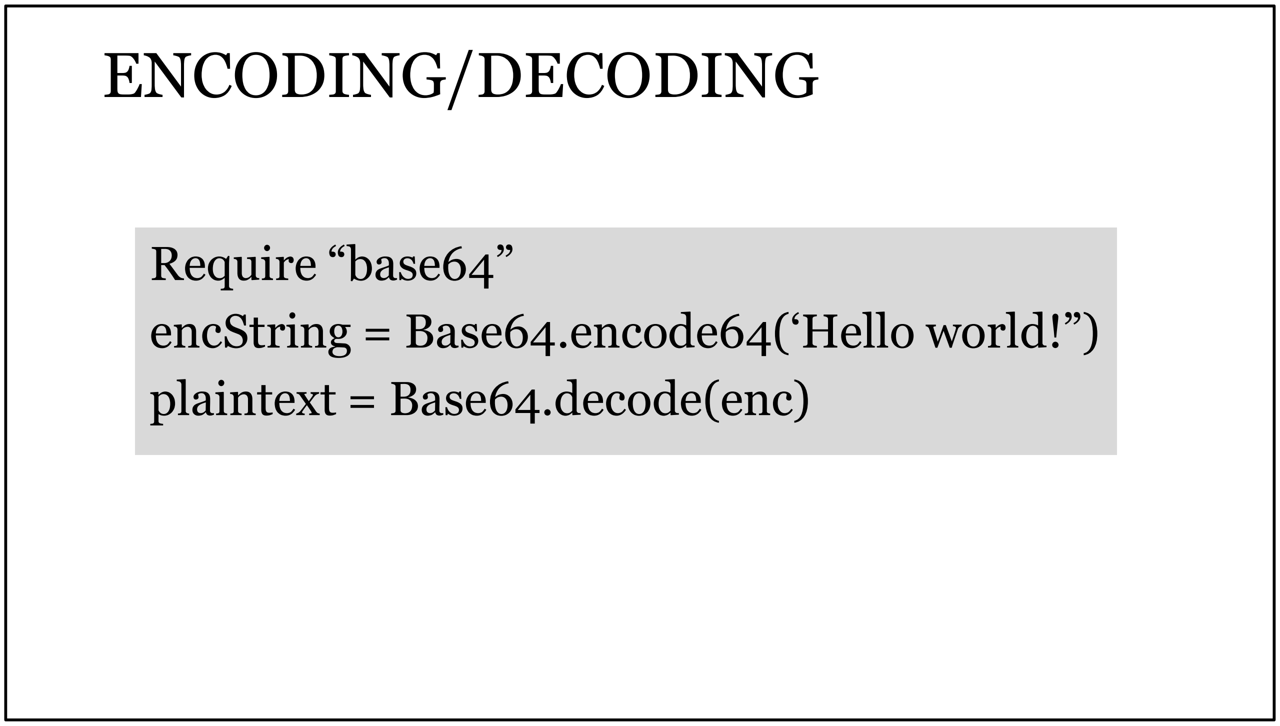
```
begin
   statements
rescue
   statements if error occurred
else
   statements if no error
end
```

4.4.6 Ruby Error handling

ARRAYS

```
rubyArray = [ “val1”, “val2”, “val3” ]
print rubyArray[1]
print rubyArray.index(“val2”)
print rubyArray.last OR print rubArray[-1]
```

4.4.7 Ruby Arrays
4.4.8 Ruby Encoding/decoding

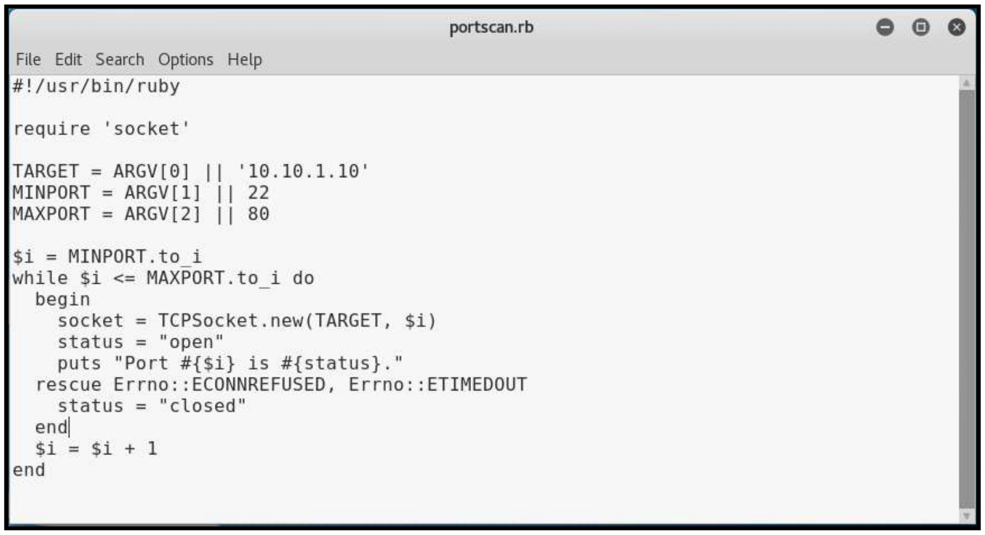


ENCODING/DECODING

```
Require “base64”
encString = Base64.encode64(‘Hello world!”)
plaintext = Base64.decode(enc)
```

4.4.7 Ruby Arrays
4.4.8 Ruby Encoding/decoding

RUBY: PUTTING IT ALL TOGETHER



```
#!/usr/bin/ruby

require 'socket'

TARGET = ARGV[0] || '10.10.1.10'
MINPORT = ARGV[1] || 22
MAXPORT = ARGV[2] || 80

$i = MINPORT.to_i
while $i <= MAXPORT.to_i do
  begin
    socket = TCPSocket.new(TARGET, $i)
    status = "open"
    puts "Port #{$i} is #{status}."
  rescue Errno::ECONNREFUSED, Errno::ETIMEDOUT
    status = "closed"
  end
  $i = $i + 1
end
```

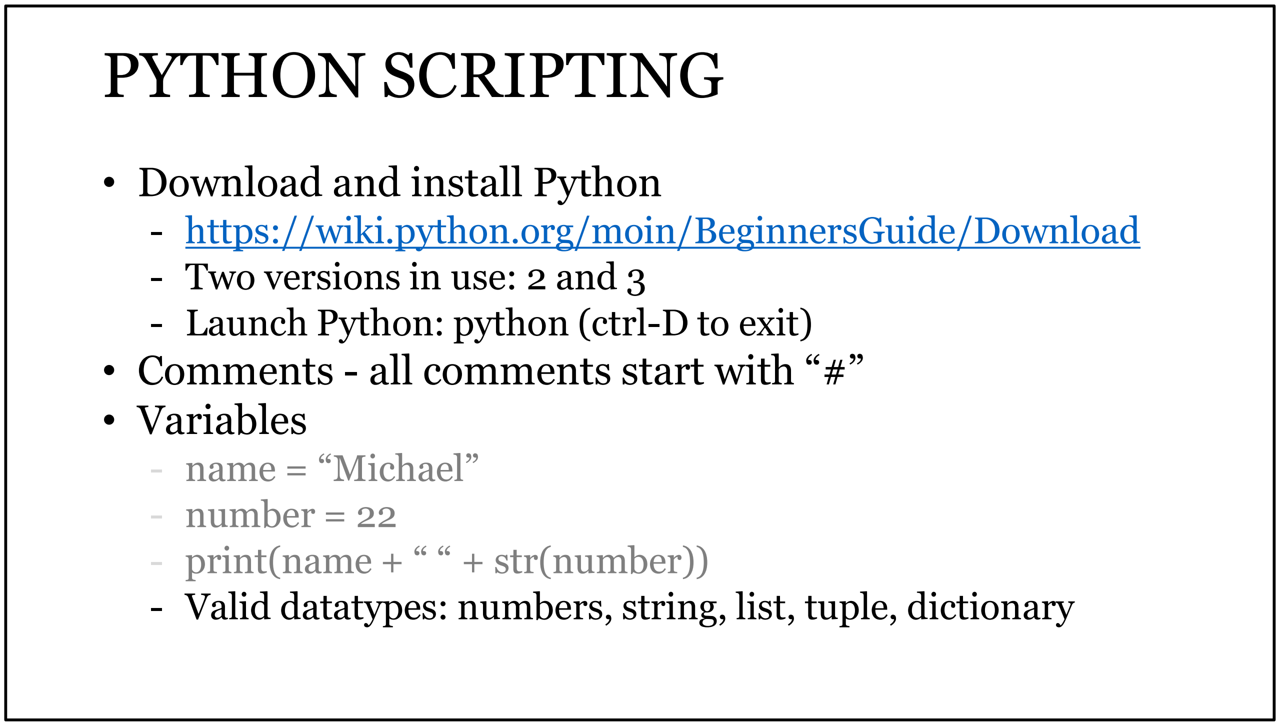
http://www.rubyguides.com/2016/11/port-scanner-in-ruby/
https://www.sitepoint.com/build-a-port-scanner-in-ruby/

```
require 'socket'

TIMEOUT = 2

def scan_port(port)
  socket      = Socket.new(:INET, :STREAM)
  remote_addr = Socket.sockaddr_in(port, 'www.example.com')

  begin
    socket.connect_nonblock(remote_addr)
  rescue Errno::EINPROGRESS
  end

  _, sockets, _ = IO.select(nil, [socket], nil, TIMEOUT)

  if sockets
```

```
    p "Port #{port} is open"
  else
    # Port is closed
  end
end

PORT_LIST = [21,22,23,25,53,80,443,3306,8080]
threads   = []

PORT_LIST.each { |i| threads << Thread.new { scan_port(i) } }

threads.each(&:join)
```

Python Scripts

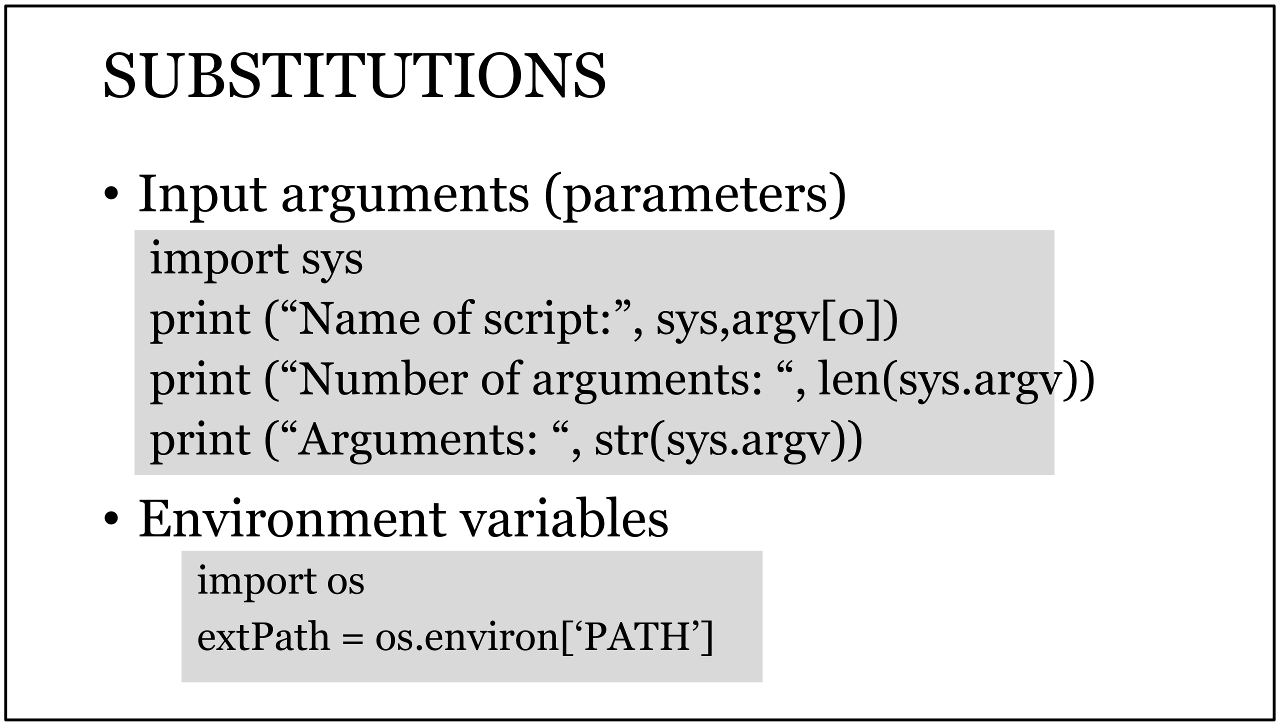
Episode 6



PYTHON SCRIPTING

- Download and install Python
  - https://wiki.python.org/moin/BeginnersGuide/Download
  - Two versions in use: 2 and 3
  - Launch Python: python (ctrl-D to exit)
- Comments - all comments start with “#”
- Variables
  - name = “Michael”
  - number = 22
  - print(name + “ “ + str(number))
  - Valid datatypes: numbers, string, list, tuple, dictionary

4.4.4 Python Variables



SUBSTITUTIONS

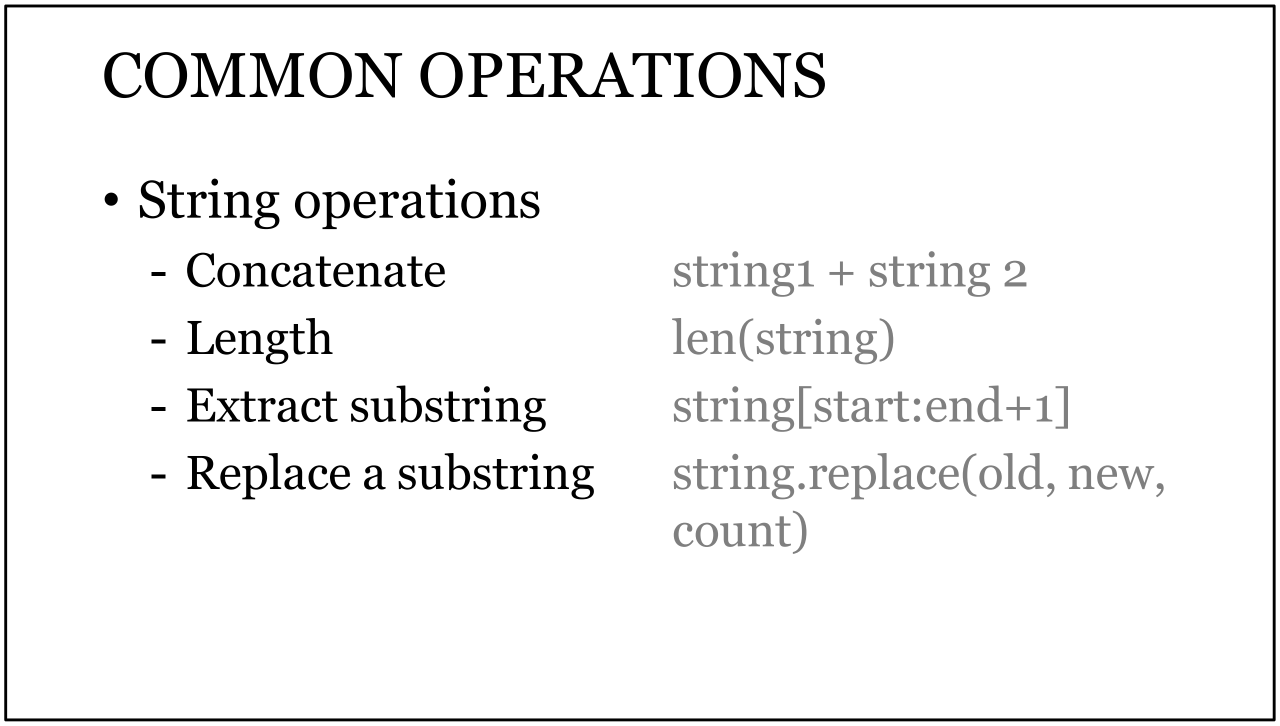
- Input arguments (parameters)

```
import sys
print (“Name of script:”, sys,argv[0])
print (“Number of arguments: “, len(sys.argv))
print (“Arguments: “, str(sys.argv))
```

- Environment variables

```
import os
extPath = os.environ[‘PATH’]
```

4.4.3 Python Substitutions

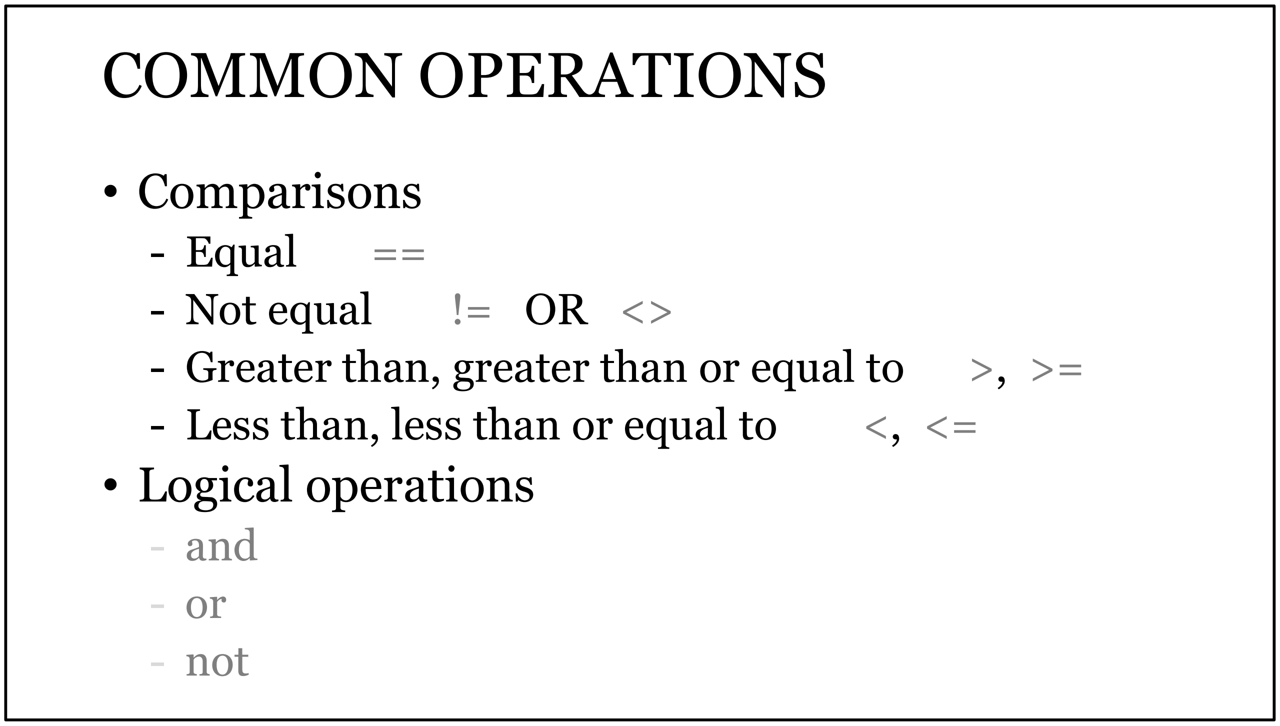


COMMON OPERATIONS

- String operations

- Concatenate	string1 + string 2
- Length	len(string)
- Extract substring	string[start:end+1]
- Replace a substring	string.replace(old, new, count)

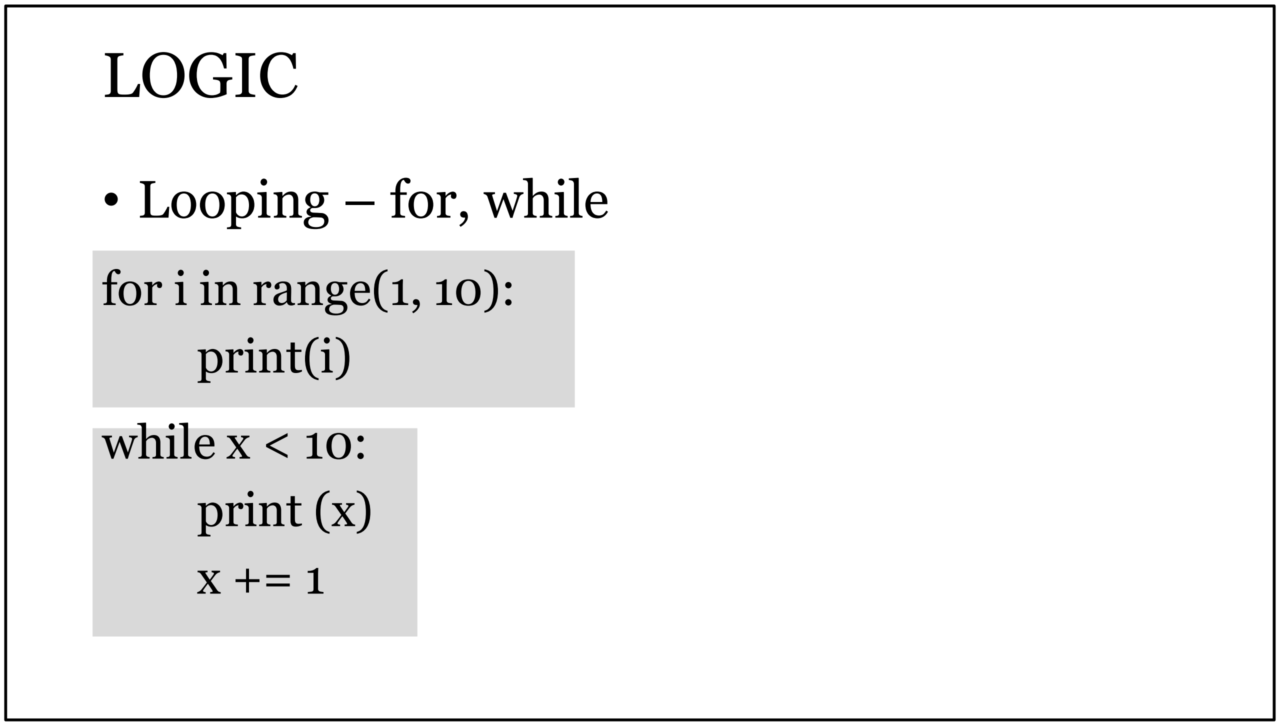
4.4.5 Python Common operations
4.4.5.1 String operations



COMMON OPERATIONS

- Comparisons
  - Equal `==`
  - Not equal `!=` OR `<>`
  - Greater than, greater than or equal to `>`, `>=`
  - Less than, less than or equal to `<`, `<=`
- Logical operations
  - and
  - or
  - not

4.4.5 Python Common operations
4.4.5.2 Comparisons



LOGIC

- Looping – for, while

```
for i in range(1, 10):
        print(i)
```

```
while x < 10:
        print (x)
        x += 1
```

4.4.1 Python Logic
4.4.1.1 Looping

LOGIC

- Flow control – if

```
if var == value:
      statements
elif var > value:
      statements
else:
      statements
```

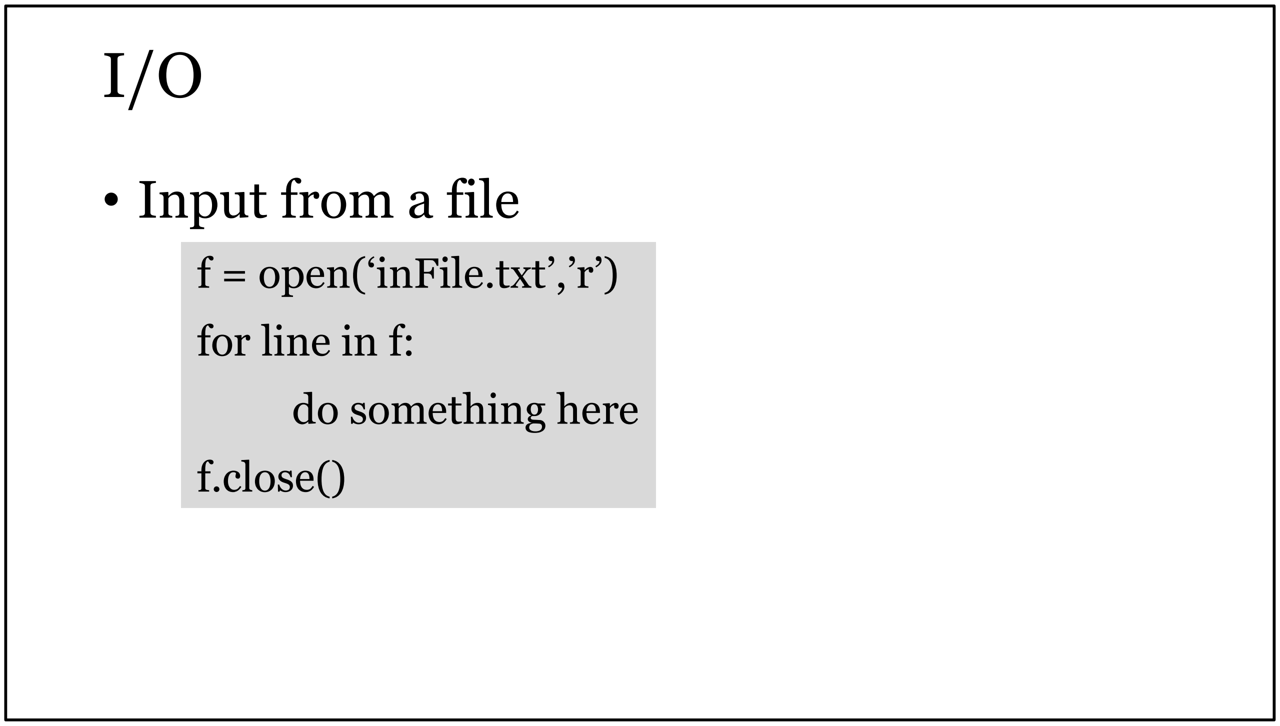
- Notice indentation

4.4.1 Python Logic
4.4.1.2 Flow control

I/O

- File vs. terminal vs. network
  - Input from a terminal
    - name = raw_input(‘Please enter your name’) # map to simple datatype
    - toppings = input(‘Which toppings do you want on your pizza?’) # maps to complex datatype
    - Input() will store data in the “best” datatype (i.e. list, etc.)

4.4.2 Python I/O
4.4.2.1 File vs. terminal vs. network



I/O

- Input from a file

```
f = open(‘inFile.txt’,’r’)
for line in f:
        do something here
f.close()
```

4.4.2 Python I/O
4.4.2.1 File vs. terminal vs. network

I/O

- Output to a file

```
f = open('outFile.txt','w')
for i in range(1,11):
        print >> f, I
f.close()
```

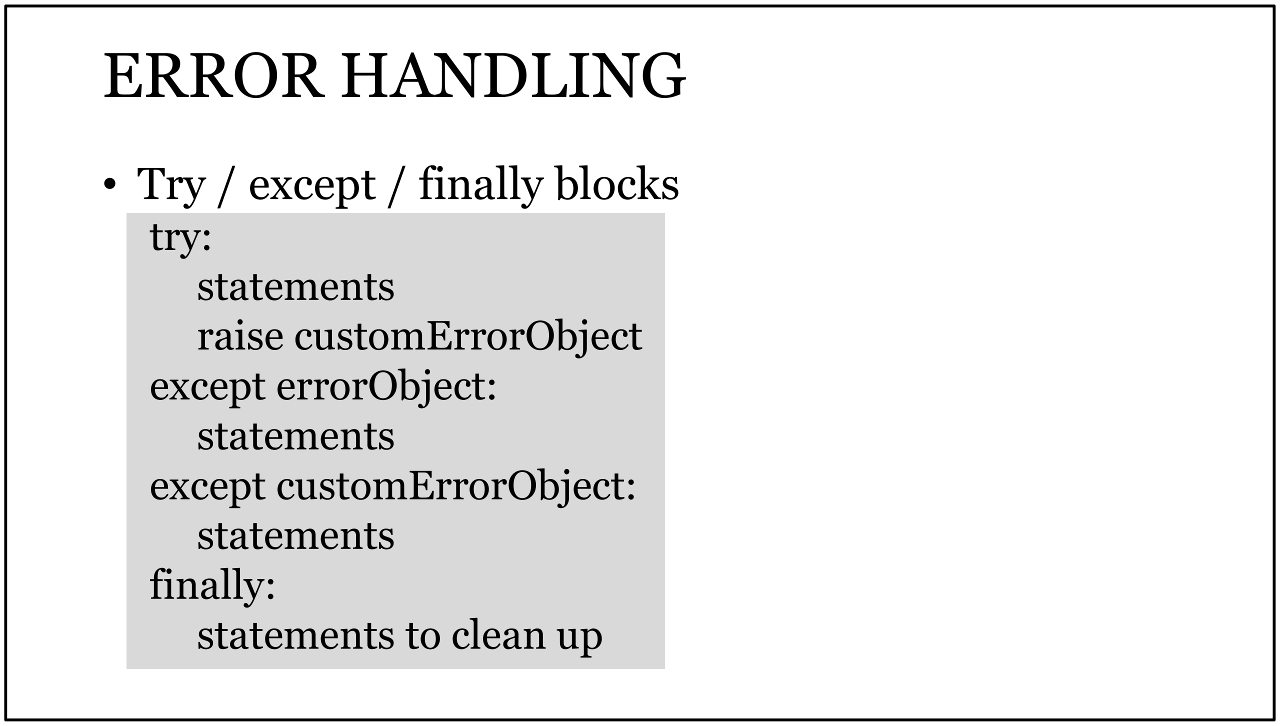
4.4.2 Python I/O
4.4.2.1 File vs. terminal vs. network

I/O

- Input from a network

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
If sock.connect_ex((remoteServerIP, port)) == 0:
    print (‘Port {}: is Open’.format(port)
```

4.4.2 Python I/O
4.4.2.1 File vs. terminal vs. network

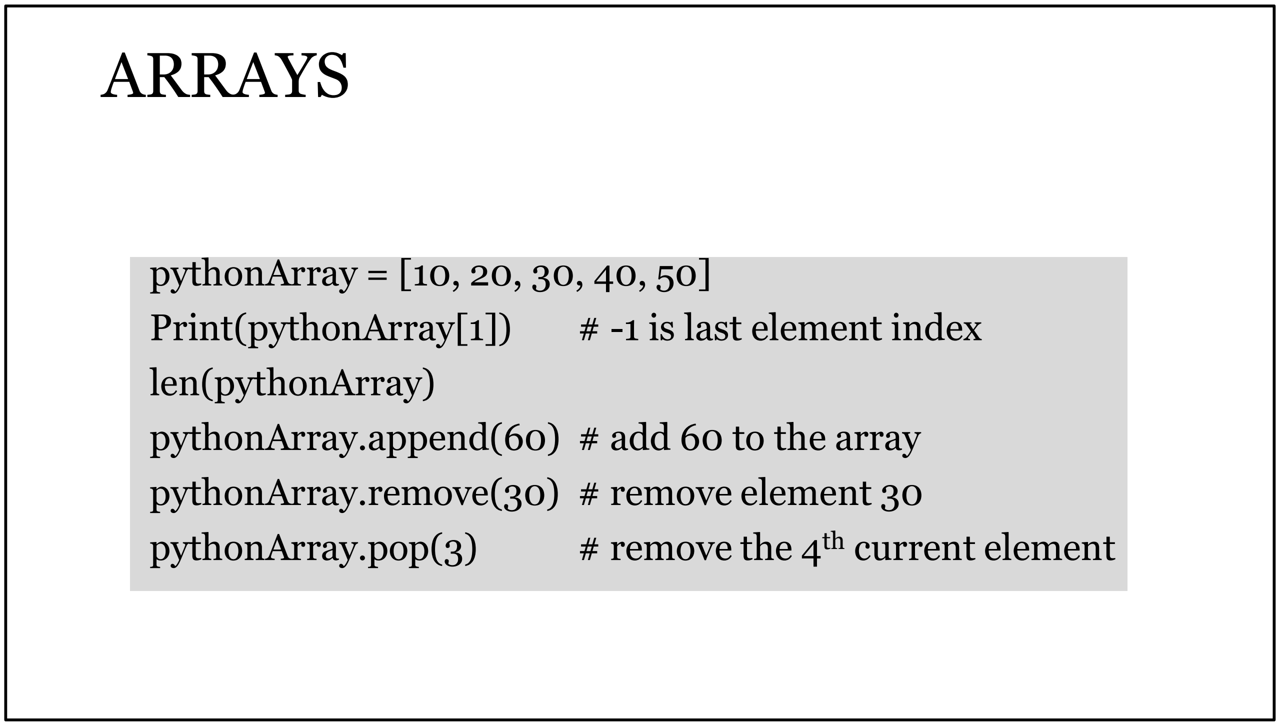


ERROR HANDLING

- Try / except / finally blocks

```
try:
    statements
    raise customErrorObject
except errorObject:
    statements
except customErrorObject:
    statements
finally:
    statements to clean up
```

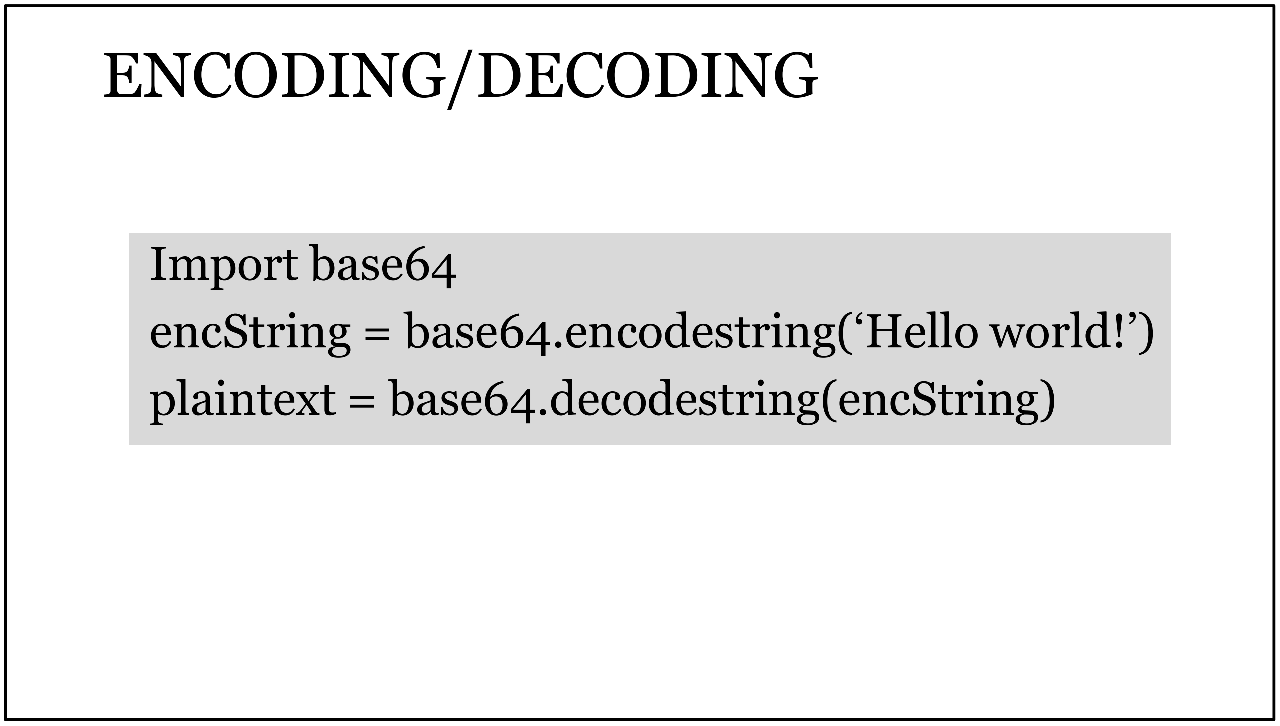
4.4.6 Python Error handling



ARRAYS

```
pythonArray = [10, 20, 30, 40, 50]
Print(pythonArray[1])      # -1 is last element index
len(pythonArray)
pythonArray.append(60)  # add 60 to the array
pythonArray.remove(30)  # remove element 30
pythonArray.pop(3)         # remove the 4th current element
```

4.4.7 Python Arrays

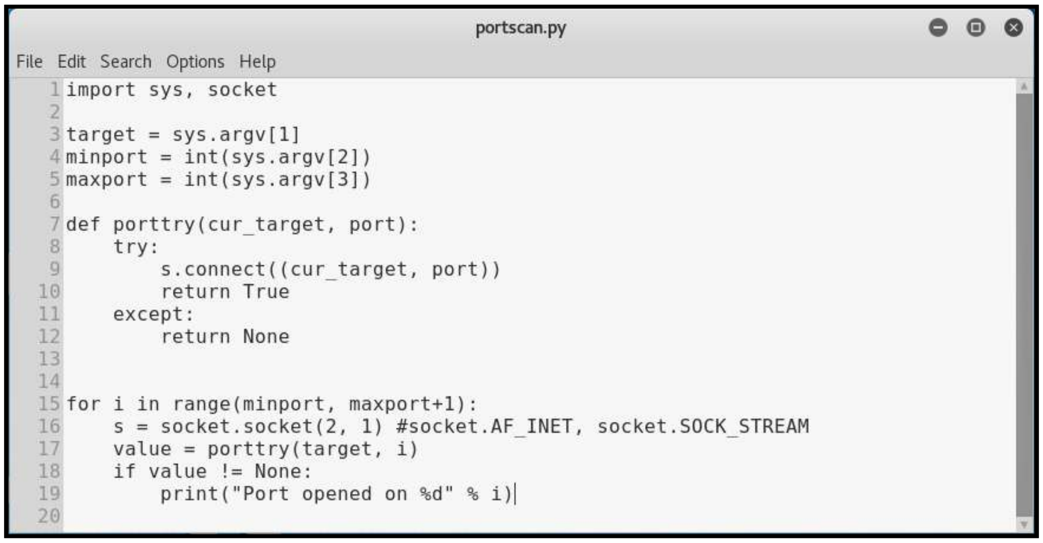


ENCODING/DECODING

```
Import base64
encString = base64.encodestring(‘Hello world!’)
plaintext = base64.decodestring(encString)
```

4.4.8 Python Encoding/decoding

Python: Putting it all together



```
import sys, socket

target = sys.argv[1]
minport = int(sys.argv[2])
maxport = int(sys.argv[3])

def porttry(cur_target, port):
    try:
        s.connect((cur_target, port))
        return True
    except:
        return None


for i in range(minport, maxport+1):
    s = socket.socket(2, 1) #socket.AF_INET, socket.SOCK_STREAM
    value = porttry(target, i)
    if value != None:
        print("Port opened on %d" % i)

```

https://gist.github.com/TheZ3ro/7255052

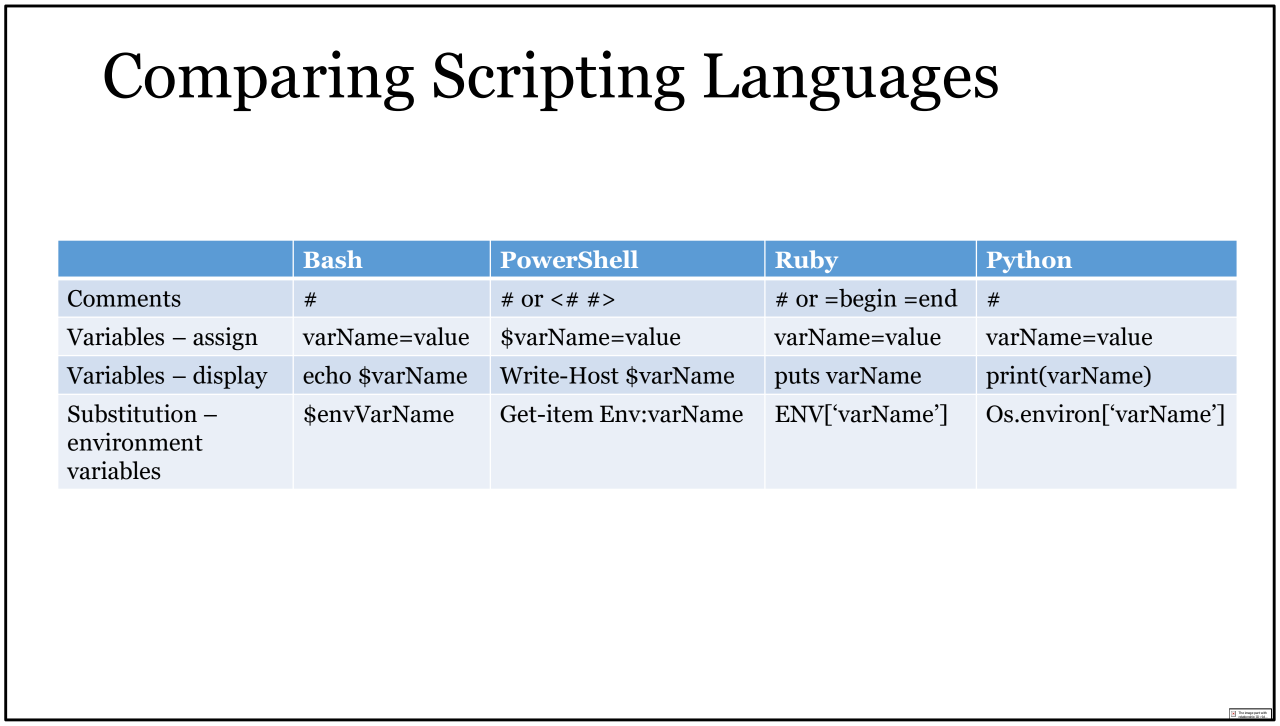
```
#!/usr/bin/env python
#TheZero
#This code is under Public Domain

from threading import Thread
import socket
host = raw_input('host > ')
from_port = input('start scan from port > ')
to_port = input('finish scan to port > ')
counting_open = []
counting_close = []
threads = []

def scan(port):
                s = socket.socket()
                result = s.connect_ex((host,port))
                print('working on port > '+(str(port)))
                if result == 0:
                                counting_open.append(port)
                                #print((str(port))+' -> open')
                                s.close()
                else:
                                counting_close.append(port)
                                #print((str(port))+' -> close')
                                s.close()

for i in range(from_port, to_port+1):
                t = Thread(target=scan, args=(i,))
                threads.append(t)
                t.start()

[x.join() for x in threads]

print(counting_open)
```

https://stackoverflow.com/questions/26174743/making-a-fast-port-scanner

```
import socket
ip = "External IP"
s = socket.socket(2, 1) #socket.AF_INET, socket.SOCK_STREAM

def porttry(ip, port):
  try:
    s.connect((ip, port))
    return True
  except:
    return None

for port in range(0, 10000):
  value = porttry(ip, port)
  if value == None:
    print("Port not opened on %d" % port)
  else:
    print("Port opened on %d" % port)
    break
raw_input()
```

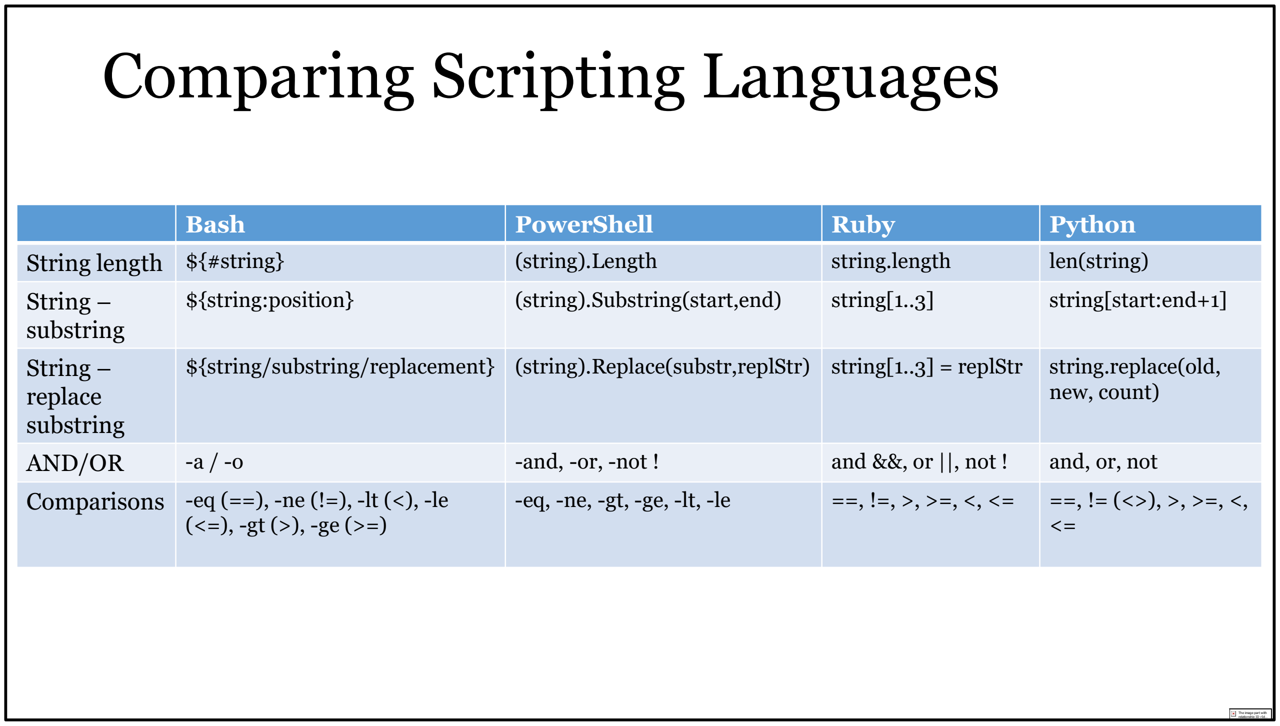
Scripting Languages Comparison

Episode 7



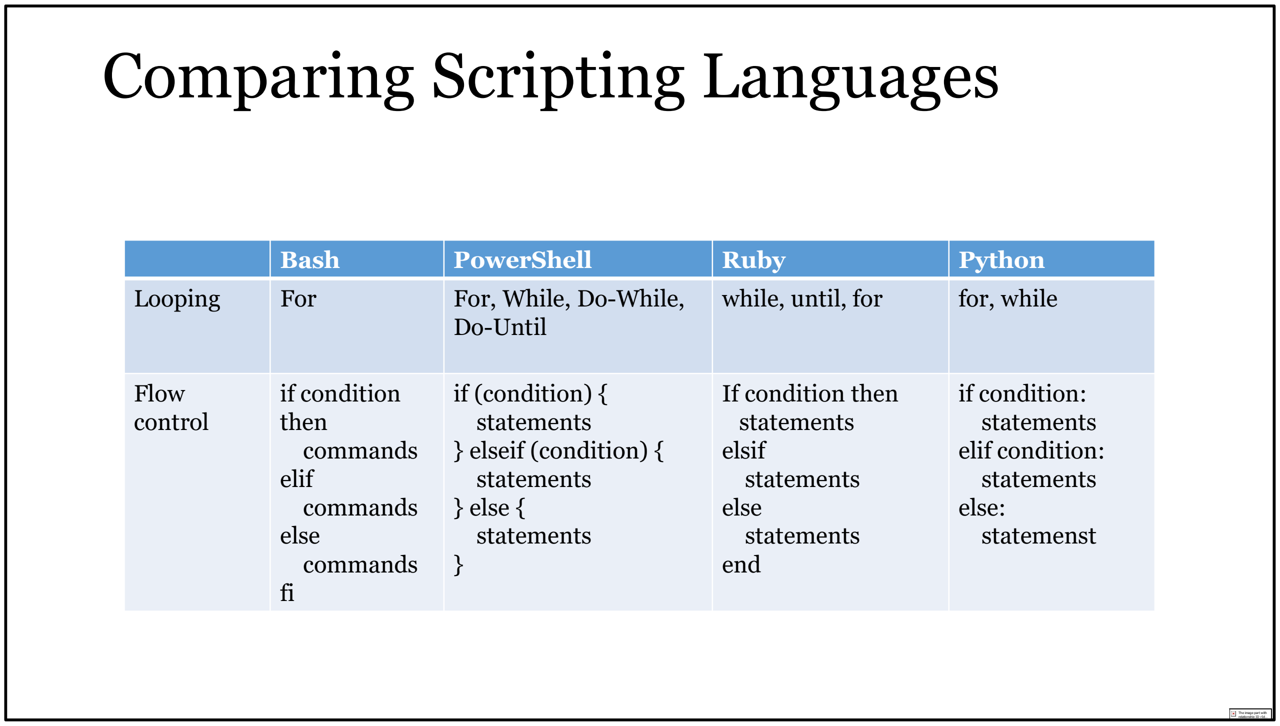
Comparing Scripting Languages

	Bash	PowerShell	Ruby	Python
Comments	#	# or <# #>	# or =begin =end	#
Variables – assign	varName=value	$varName=value	varName=value	varName=value
Variables – display	echo $varName	Write-Host $varName	puts varName	print(varName)
Substitution – environment variables	$envVarName	Get-item Env:varName	ENV[‘varName’]	Os.environ[‘varName’]



Comparing Scripting Languages

	Bash	PowerShell	Ruby	Python
String length	${#string}	(string).Length	string.length	len(string)
String – substring	${string:position}	(string).Substring(start,end)	string[1..3]	string[start:end+1]
String – replace substring	${string/substring/replacement}	(string).Replace(substr,replStr)	string[1..3] = replStr	string.replace(old, new, count)
AND/OR	-a / -o	-and, -or, -not !	and &&, or ||, not !	and, or, not
Comparisons	-eq (==), -ne (!=), -lt (<), -le (<=), -gt (>), -ge (>=)	-eq, -ne, -gt, -ge, -lt, -le	==, !=, >, >=, <, <=	==, != (<>), >, >=, <, <=



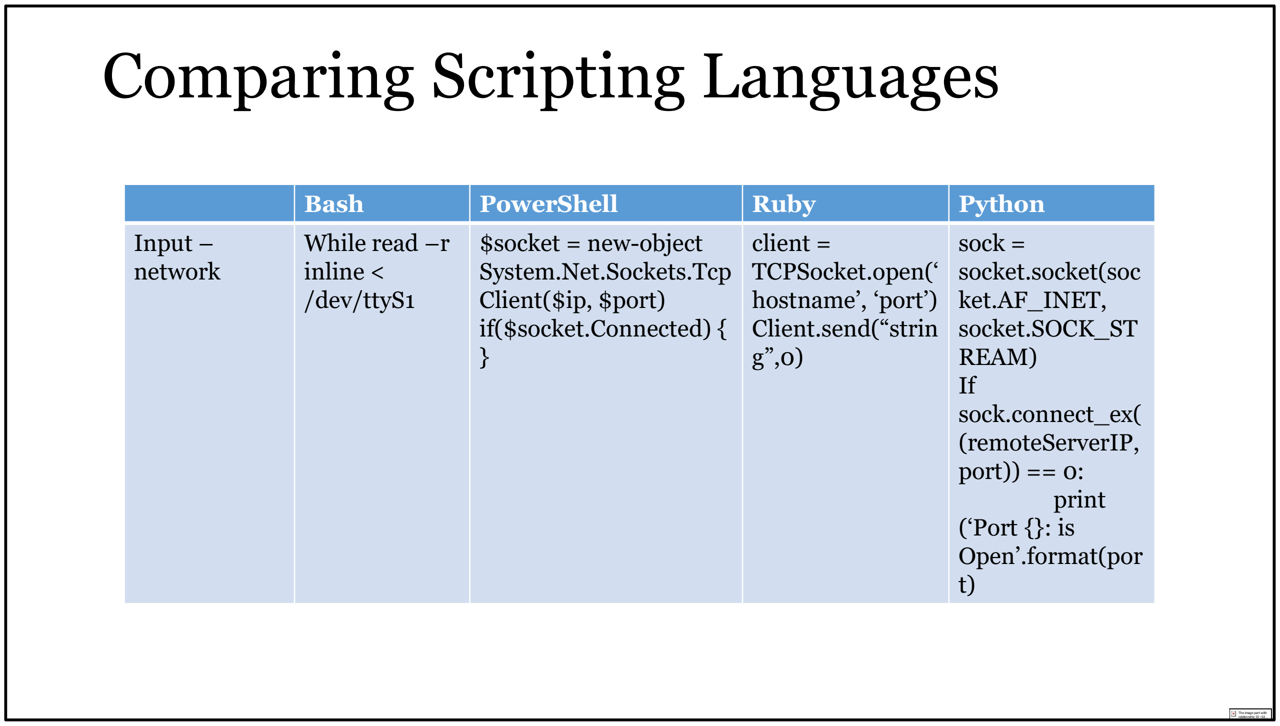
Comparing Scripting Languages

	Bash	PowerShell	Ruby	Python
Looping	For	For, While, Do-While, Do-Until	while, until, for	for, while

Flow control	if condition
	then
	commands
	elif
	commands
	else
	commands
	fi
		if (condition) {
		statements
		} elseif (condition) {
		statements
		} else {
		statements
		}
			If condition then
			statements
			elsif
			statements
			else
			statements
			end
				if condition:
				statements
				elif condition:
				statements
				else:
				statenenst

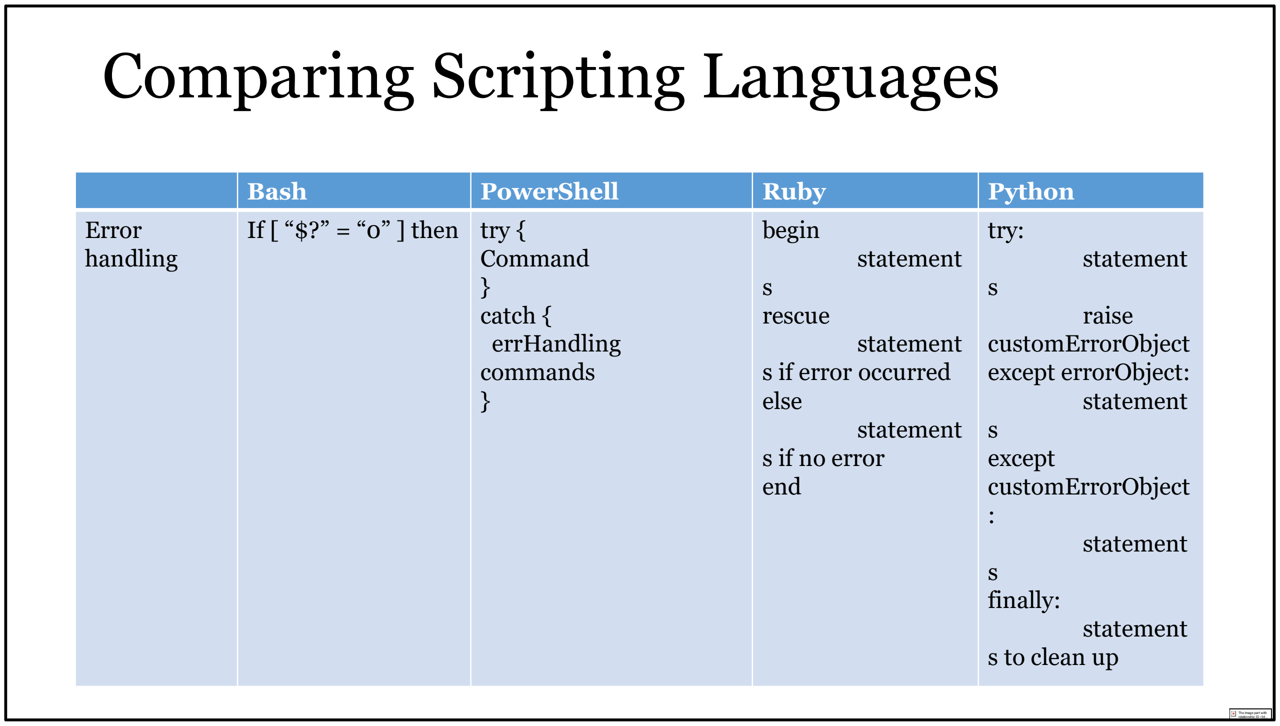
Comparing Scripting Languages

	Bash	PowerShell	Ruby	Python
Input – file	Input=“filename” While IFS=read –r f1 f2 f3	$lines = Get-Content filename Out-File –FilePath filename –InputObject $lines –Encoding ASCII	inFile = File.new(“filename”,”r”) inFile.each_line {|line| puts “#{line.dump}” } inFile.close	f = open(‘inFile.txt’,’r’) for line in f: do something here f.close()
Input – terminal	Read –p “Prompt:” var	$firstName = Read-Host –Prompt ‘Enter first name’	name = gets	name = raw_input(‘Please enter your name’)



Comparing Scripting Languages

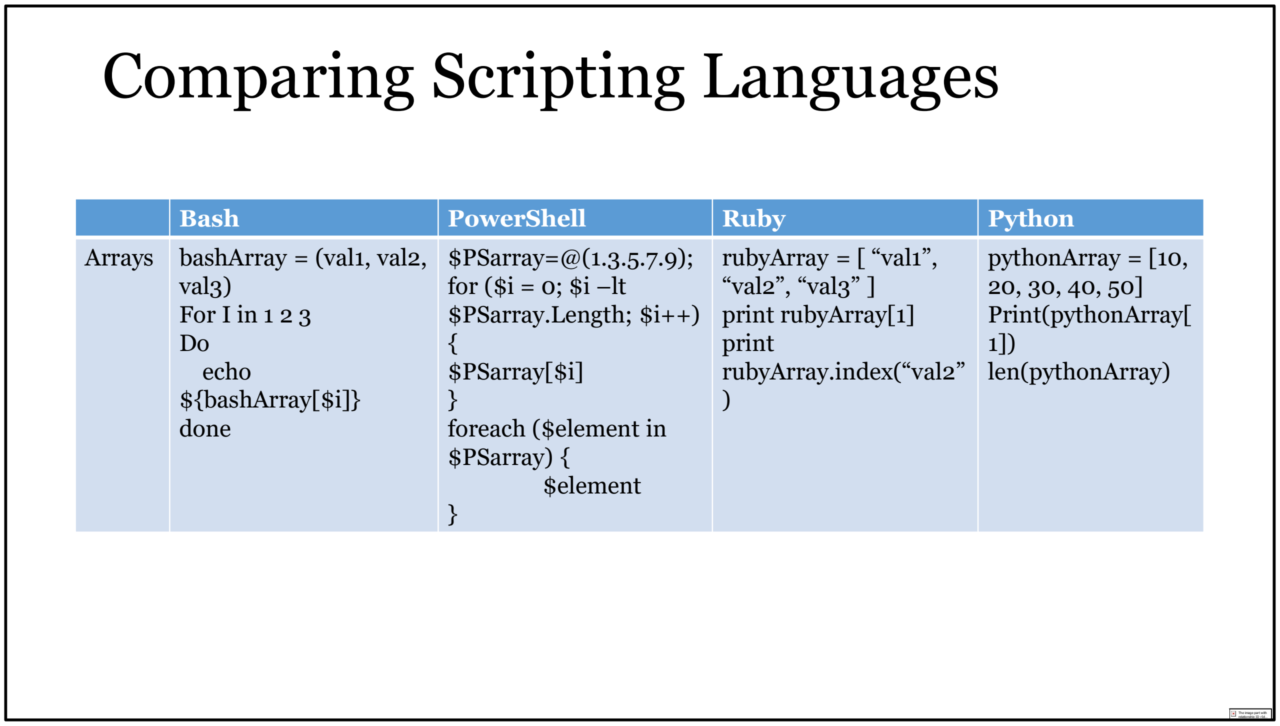
	Bash	PowerShell	Ruby	Python
Input – network	While read –r inline < /dev/ttyS1	$socket = new-object System.Net.Sockets.TcpClient($ip, $port) if($socket.Connected) { }	client = TCPSocket.open('hostname', 'port') Client.send("string",0)	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) If sock.connect_ex((remoteServerIP, port)) == 0: print ('Port {}: is Open'.format(port)



Comparing Scripting Languages

	Bash	PowerShell	Ruby	Python

Error handling	If ["$?" = "0"] then	try {
		Command
		}
		catch {
		errHandling commands
		}
			begin
			statements
			rescue
			statements if error occurred
			else
			statements if no error
			end
				try:
				statements
				raise customErrorObject
				except errorObject:
				statements
				except customErrorObject:
				statements
				finally:
				statements to clean up



Comparing Scripting Languages

	Bash	PowerShell	Ruby	Python

Arrays	bashArray = (val1, val2, val3)
	For I in 1 2 3
	Do
	   echo ${bashArray[$i]}
	done
		$PSarray=@(1.3.5.7.9);
		for ($i = 0; $i –lt $PSarray.Length; $i++)
		{
		$PSarray[$i]
		}
		foreach ($element in $PSarray) {
		        $element
		}
			rubyArray = [“val1”, “val2”, “val3”]
			print rubyArray[1]
			print rubyArray.index(“val2”)
				pythonArray = [10, 20, 30, 40, 50]
				Print(pythonArray[1])
				len(pythonArray)

Comparing Scripting Languages

	Bash	PowerShell	Ruby	Python
Encoding	Echo plainText | base64	$Text = ‘Hello world’ $Bytes = [System.Text.Encoding]::Unicode.GetByteps($Text) $EncodedText = [Convert]::ToBase64String.($Bytes	Require “base64” encString = Base64.encode64(‘Hello world!”)	Import base64 encString = base64.encodestring(‘Hello world!’)
Decoding	Echo encString | base64 --decode	$EncodedText = ‘encodedString’ $DecodedText = [System.Tet.Encoding]::Unicode.GetString([System.Convert]::FromBase64String($EncodedText)	plaintext = Base64.decode(enc)	plaintext = base64.decodestring(encString)