LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

PYTHON

CNO CORE MODULE

ManTech Advanced Systems International, Inc.
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ManTech

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Series Overview

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ManTech

ACTP
ADVANCED CYBER TRAINING PROGRAM

PYTHON

SERIES

- Series Outcome
  - After finishing the Python Series, the student will be familiar with the syntax and common commands of the language, as well as be able to create a basic program
  - Python is one of the languages used by CNO Developers and is an important development tool
- Assessments
  - Daily Quizzes and Labs (don't count towards grade)
  - Final Assessment comprised of multiple-choice questions:
    - Questions contingent upon the successful completion of integrated lab(s)
    - Minimum score of 80% to pass

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019
ManTech

Learning Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Use the Python Interpreter to complete basic tasks
- Use basic types and operators in Python
- Use basic statements in Python
- Write a basic function
- Describe the use of generator expressions as well as functions as objects
- Create and use modules in Python
- Create and use classes in Python
- Develop a custom exception class
- Demonstrate basic use of Python built-in tools
- Use available system interfaces in Python

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

ACTP
ADVANCED CYBER TRAINING PROGRAM

Python Series Agenda

- Python Interpreter
- Types and Operators
- Statements
- Functions
- Modules/Packages
- Classes
- Exceptions
- Built-in Tools
- System Interfaces

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019
ManTech

ACTP
ADVANCED CYBER TRAINING PROGRAM

REFERENCES

- Online References
  - http://docs.python.org
  - https://docs.python.org/3/tutorial/
  - https://www.w3schools.com/python/
  - http://stackoverflow.com/questions/tagged/python
  - https://developers.google.com/edu/python
- Challenges
  - http://www.pythonchallenge.com/
  - https://www.practicepython.org/
  - https://wiki.python.org/moin/ProblemSets

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Python Mini Crucible

Sneak Peak...

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ACTP
ADVANCED CYBER TRAINING PROGRAM

PyCTF

Sneak Peak...

ADVANCED CYBER TRAINING PROGRAM

Background

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019
ManTech

What is Python?

- "An Interpreted, Object Oriented, High-level programming language with dynamic semantics."
- Interpreted – no compilation
  - Executing the text
  - Edit-Test-Debug cycle is fast
  - Debugging Python is easy – bugs and bad input cause "exceptions" – (nearly) never cause segmentation faults
  - Exceptions, when un-caught, print a "stack trace"
  - Python built-in debugging API is available (*pdb*)

https://www.python.org/doc/essays/blurb/

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

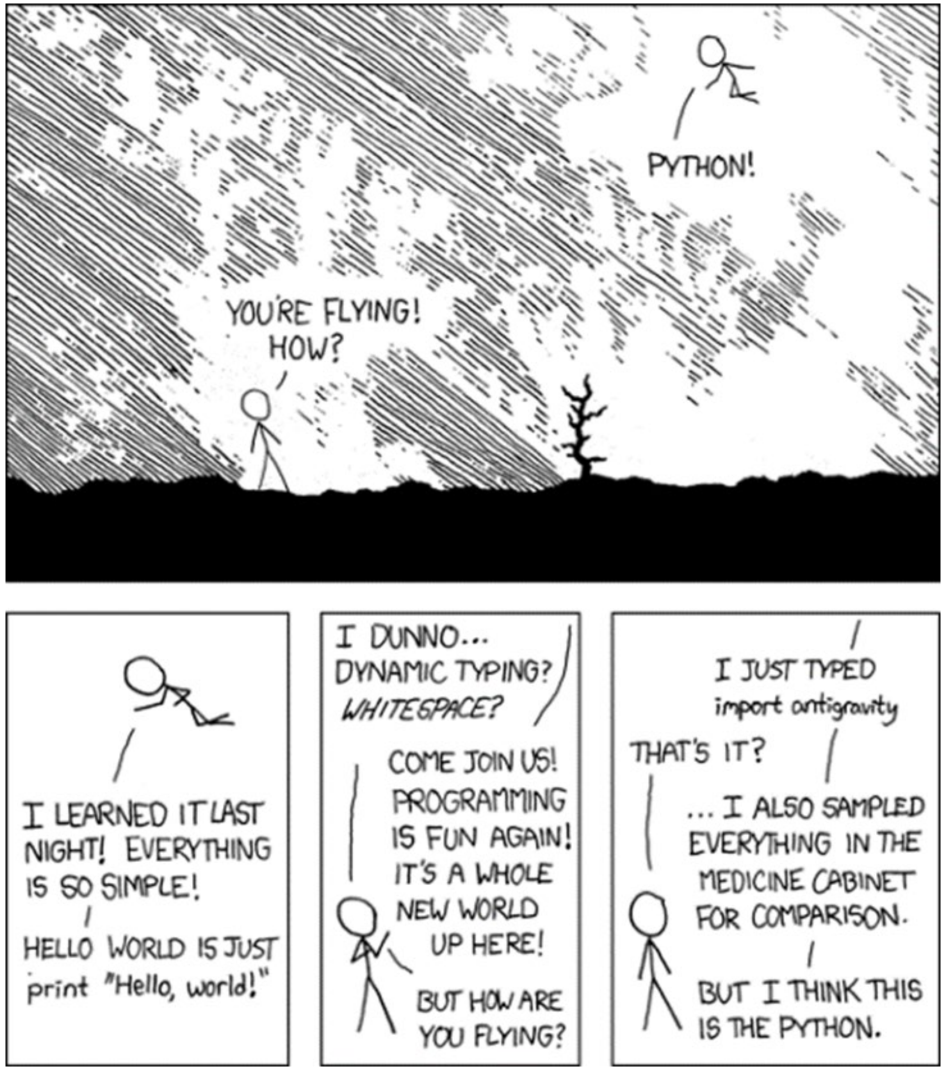
Why Do We Use Python?

- Simple, easy to learn syntax emphasizes readability
- Supports modules and packages, encouraging modularity and code reuse
- Interpreter and standard libraries are available for all major platforms and can be freely distributed
- Philosophy of:
  - "There should be one -- and preferably only one -- obvious way to do it." - Guido van Rossum

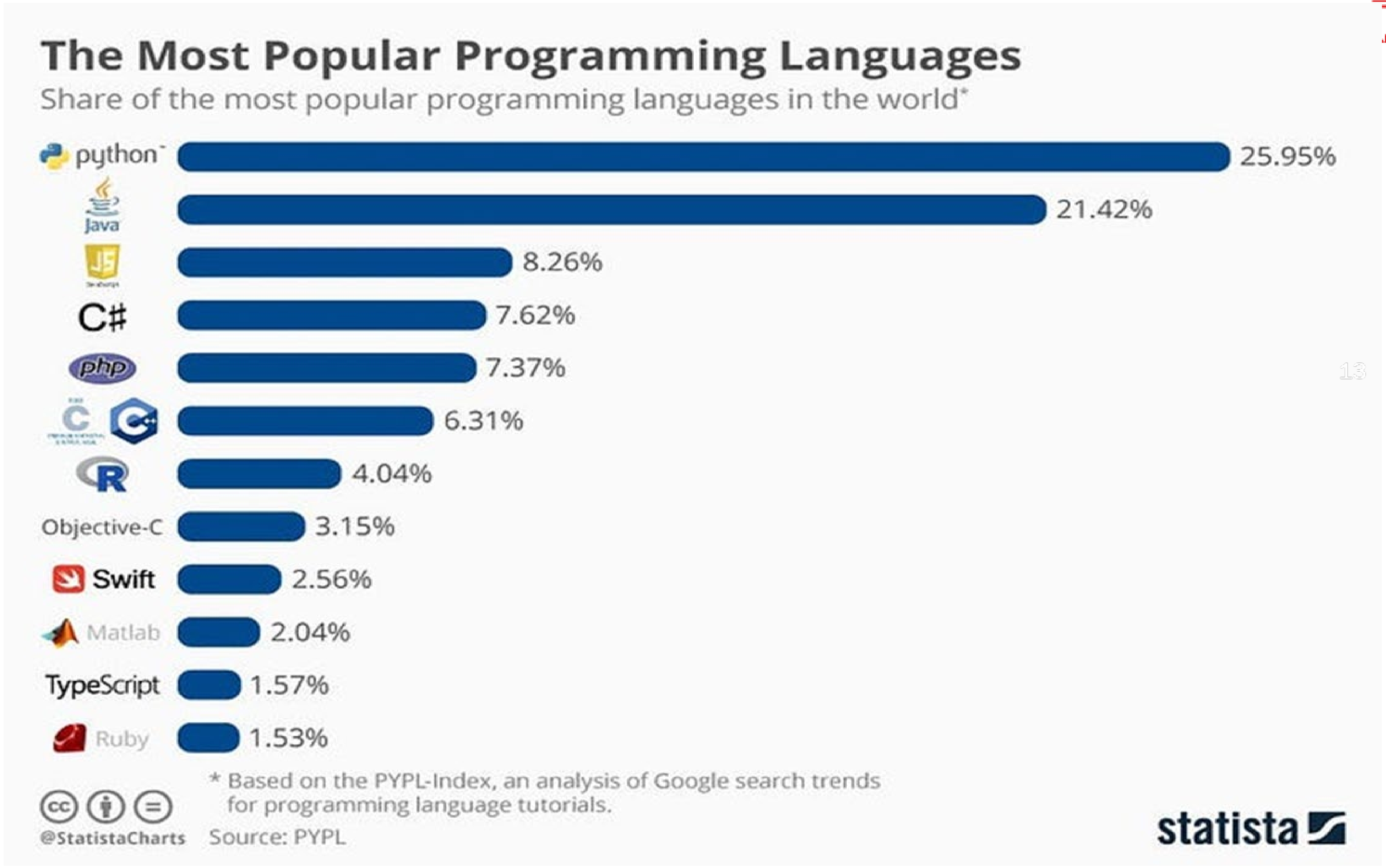
```
>>> import this
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

import antigravity


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019



Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

The History of Python

- Created in the early 1990's by Guido Van Rossum
- Python 2.0 was released on October 16, 2000
- Python 3.0 was released on December 3, 2008
  - Not completely backwards compatible. Many major features backported to Python 2.6.x and 2.7.x
- Python 2 (2.7.x) was officially discontinued on January 1, 2020
- Active Python Releases https://python.org/downloads/

	Version	Release	End of support	Release schedule
	3.13	2024-10-01*	2029-10	PEP 719
-->	3.12	2023-10-02	2028-10	PEP 693
	3.11	2022-10-24	2027-10	PEP 664
	3.10	2021-10-04	2026-10	PEP 619
	3.9	2020-10-05	2025-10	PEP 596
	3.8	2019-10-14	2024-10	PEP 569

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

When To Use Python

- Simple scripting
- "Gluing" other software together
- Prototyping
- Rapid Application Development
- Cross platform applications
- Unit testing

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ADVANCED CYBER TRAINING PROGRAM

The Python Interpreter

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objectives
  - use the Python Interpreter to complete basic tasks

- Enabling Objectives
  - Use the Python interactive interpreter to develop code snippets
  - Describe how a user runs a Python script
  - Describe how Python module files are used

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Interactive Interpreter

- A command line like environment
- Using a standard REP Loop (**REPL**)
  - **R**ead
    - Read in one Python expression
    - Ex: >>> 7+6
  - **E**valuate
    - Compute the result of the expression
    - 7+6 = 13
  - **P**rint
    - 13
  - **L**oop
    - Returns to the standard Python Interpreter prompt: **>>>**

ManTech

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Interactive Interpreter, continued

Lots of information is available from the Python Interpreter

- Version (3.7.7)
- Build Date (13 March 2020)
- 64 bit Python

```
[/usr/bin]$ python3
Python 3.7.7 (default, Mar 13 2020, 21:39:43)
[GCC 9.2.1 20190827 (Red Hat 9.2.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 7 + 6
13
>>>
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Running Python Scripts

- Generally, running a Python script involves one of two options:
  - Calling the Python interpreter on the command line with the name of the script you want to run as an argument
    - C:\Python38\python.exe hw.py ->Windows
    - /usr/bin/python hw.py -> Linux
  - Making the script executable and embedding a directive to "run" as the first line of the script
    - On Windows the Python installer *usually* sets the Python binary to be the default program to run when a .py file is invoked
- Often a Python module/package can be imported, and then sub-components can be run/tested individually

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Using Modules

- Modules are:
  - Files containing Python definitions and statements
  - Useful for saving commonly used code chunks
  - Necessary for larger applications
- Using modules is easy!

```
# file:foo.py
print("x = 3")          # assign 3 to x
x = 3
print("y = 10")         # assign 10 to y
y = 10
print("x + y")          # operation
print(x + y)
```

```
>>> import foo
x = 3
y = 10
x + y
13
>>>
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python Docs

- F1 from within IDLE
- Start/Search -> Python X.X -> Python Manuals
- HTML documentation can be obtained online, and allows for searching the docs in your preferred browser
- Can give you lots of info on Python keywords / built-ins / functions
- Will become your best friend for the next couple of days / the rest of your life

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python Self-Help

- When using the interpreter, useful information can be obtained by using: dir() and help()

```
>>> x = 3
>>> dir(x)
[ ...(many private functions)..., 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag',
'numerator', 'real', 'to_bytes']
>>> help(x.to_bytes)
Help on built-in function to_bytes:

to_bytes(length, byteorder, *, signed=False) method of builtins.int instance
    Return an array of bytes representing an integer.

...(lots of information regarding the parameters and how they are used)...
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *pdb*

- Python pdb module:
  - Built-in, interactive source code debugger for Python
    - import pdb
  - Very handy when print statements don't cut it
  - Allows you to break into debugger from a running program
    - pdb.set_trace()
      - breakpoint() (3.7+)
    - Can inspect variables and program state
  - See Python docs for more info!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Modules/Packages

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objectives
  - Use modules and packages in Python.
- Enabling Objectives
  - Describe why modules/packages are used
  - Describe how modules/packages are created and used
  - Describe the module/package search order and how to modify it
  - Demonstrate how modules/packages are used as objects
  - Generate multiple namespaces using modules/packages

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

What are Modules?

- Reusable code intended for use across multiple Python scripts or programs
- Used for structuring code, allowing easy maintenance and logical organization
- Single files containing Python definitions and statements
- Modules may be any of the following file types:
  - source (".py")
  - byte-code (".pyc")
    - __pycache__
  - Windows dll (".pyd")

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Module Structure

- A module name is the filename with the extension ".py"
- The following example is a "fibonacci" module

```
# fibonacci.py
import sys
FIB_MAX = 1000
print('Inside the fibonacci module!')
def fib(n):
    """Return Fibonacci series up to n (max FIB_MAX) as a list"""
    n = min(n, FIB_MAX)
    results = []
    a = 0
    b = 1
    while b < n:
        results.append(b)
        a, b = b, a + b

    return results

def fib2(n):
    """Print Fibonacci series up to n"""
    print(' '.join([str(k) for k in fib(n)]))

if __name__ == '__main__':
    if len(sys.argv) != 2:
        print('USAGE: %s <maxnum>' % (sys.argv[0]))
        sys.exit(-1)
    fib2(int(sys.argv[1]))
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Module/Package Use

- When a module/package is imported, the code inside that module is run, all function definitions are added to the namespace, and any variables are set

```
>>> import fibonacci
Inside the fibonacci module!
>>> fibonacci.fib(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibonacci.fib2(100)
1 1 2 3 5 8 13 21 34 55 89
>>> fibonacci.FIB_MAX
1000
```

- FIB_MAX may be overridden at runtime by assigning a new value to it

```
>>> fibonacci.FIB_MAX = 2**16
>>> fibonacci.FIB_MAX
65536
>>>
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Module/Package Use *(continued)*

- When a module/package is imported, Python sets a global called "__name__" to the name of the module/package

```
>>> import fibonacci
Inside the fibonacci module!
>>> fibonacci.__name__
'fibonacci'
```

- The following code from the fibonacci module doesn't get run on import because the name is "fibonacci"

```
if __name__ == '__main__':
    if len(sys.argv) != 2:
        print('USAGE: %s <maxnum>' % (sys.argv[0]))
        sys.exit(-1)
    fib2(int(sys.argv[1]))
```

- "__name__" is only set to "__main__" when the module is run from the command line

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

What are Packages?

- A logical collection of modules is known as a package
- Packages can also contain additional packages
- Packages are mostly intended for organization of code
- The following package includes the modules "util.py", "rmi.py", and the package "archs" which contains the module "intel.py"

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Package Structure

- The following directory listing shows package structure for the package named vtrace

```
vtrace/
    __init__.py
    util.py
    rmi.py
    archs/
        __init__.py
        intel.py
```

- A package may contain modules or other packages
- The special module "__init__.py" is required* to make Python treat a directory as a package
  - May contain initialization code or be an empty file
  - *Implicit Namespace Packages

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Package Use

- The following code example shows how to import modules from a package

```
>>> import vtrace                  # Import the vtrace package
>>> import vtrace.rmi as v_rmi     # Import the rmi module from inside the vtrace package and
                                    # assign that module to the name "v_rmi"
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Package Use *(continued)*

- Inside of **__init__.py**, the **__all__** list must be defined for "from <package> import *" to import submodules
- The "vtrace" package defines **__all__** as a list including "util", "rmi", and "archs"

```
## vtrace's __init__.py
__all__ = ['util', 'rmi', 'archs']
```

```
>>> import vtrace
>>> vtrace.__all__
['util', 'rmi', 'archs']
```

- The result from "from vtrace import *" can be seen below

```
>>> from vtrace import *
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'archs', 'rmi', 'util']
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Importing Modules/Packages

- The import function provides many ways to import a module and/or package and its contents
  - import <module/package>

```
>>> import fibonacci
>>> fibonacci
<module 'fibonacci' from 'fibonacci.py'>
```

  - import <module/package> as <altname>

```
>>> import fibonacci as fib
>>> fib
<module 'fibonacci' from 'fibonacci.py'>
```

  - from <module/package> import <object>

```
>>> from fibonacci import fib
>>> fib
<function fib at 0x100477b90>
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Importing Modules/Packages *(continued)*

- from <module/package> import <object>,<object>...

```
>>> from fibonacci import fib, fib2
>>> fib
<function fib at 0x100477b90>
>>> fib2
<function fib2 at 0x10047b0c8>
```

- from <module/package> import *

```
>>> from fibonacci import *
>>> dir()
['FIB_MAX', '__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__', 'fib', 'fib2', 'sys']
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Browsing Modules/Packages

- In the interpreter, the help() function will show pydoc formatted information about that module

```
>>> import fibonacci
>>> help(fibonacci)
Help on module fibonacci:

NAME
fibonacci - # fibonacci.py

FILE
    /Users/blackout/Downloads/fibonacci.py

FUNCTIONS
fib(n)
        Return Fibonacci series up to n as a list


fib2(n)
        Print Fibonacci series up to n

DATA
    FIB_MAX = 1000
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Browsing Modules/Packages *(continued)*

- The dir() function also provides useful information for browsing the contents of a module/package/object

```
>>> dir(fibonacci)
['FIB_MAX', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'fib',
'fib2', 'sys']
>>> fibonacci.fib
<function fib at 0x100477b90>
>>> fibonacci.fib2
<function fib2 at 0x10047b0c8>
>>> fibonacci.FIB_MAX
1000
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Module/Package Search Order

- List of directories in sys.path is searched
  - By default, sys.path is initialized to (in order):
    - Current directory
    - Directories specified in environment variable PYTHONPATH
    - Installation-dependent default path
- sys.modules dictionary contains modules that have been imported
- **Note:** If a directory contains a package and a module of the same name, Python will load the package, not the module

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Lab 1

"Hello World"

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- Open the Python docs to find out what "print" and "input" do
- Launch IDLE and output "Hello World!"
- Run Python from inside a command prompt and output "Hello World!"
- Create hw.py (using IDLE, notepad++, vim, PyCharm, Atom, VSCode, or your text editor of choice)
  - Experiment with "print" and "input"
- Run the Python executable, passing it the path to your script
- How does pathing impact your ability to launch the interpreter and Python scripts?
- Update your path variable so that you can launch Python from anywhere

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Use the Python interactive interpreter to its fullest
  - It is a great tool for rapid development
  - Provides an easy mechanism for quick testing of code snippets
- There is a lot going on behind the scenes when you invoke a Python script
- Use modules as much as possible – if you need to do it in Python, chances are someone has/will as well
- Questions?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Lab 2

Modularization

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- Create the vtrace package and modules within from slide 31
- Add a few of the methods from the previous lab to each of your modules, and experiment with *import* and *from* <> *import*
- Add your hw.py module to a new subfolder called 'my_mods' within the vtrace package/directory
- Example output:

```
vtrace/
        rmi.py
        utils.py
        my_mods/
                hw.py
        archs/
```

- What import statements enable your hw.py to run/print?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Modules are reusable code intended for use across multiple Python scripts or programs
- Modules are single files containing Python definitions and statements
- A module name is a filename with a ".py" extension
- Packages are logical collections of modules – primarily intended for code organization
- Packages may contain other packages
- Packages must contain the special module "__init__.py"
- Use help(<name>) or dir(<name>) to learn more about imported packages/modules/objects

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ADVANCED CYBER TRAINING PROGRAM

Types and Operators

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Given a workstation and technical documentation, the student will be able to use basic types and operators in Python.
- Enabling Objectives
  - Describe variables, values, and keywords
  - Describe various Python built-in types
  - Describe Python expressions
  - Describe Python bitwise operators
  - Explain precedence among Python operators
  - Use various Python built-in types
  - Describe general Python object properties
  - Use various operations

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Variables, Values, and Types

- Values are basic building blocks programs work with – letters, numbers, etc – seen previously as "x+y", 10, 3
- Values belong to different types:
  - 10 and 3 are of type integer (<class 'int'>)
  - "x+y" is of type string (<class 'str'>)
  - The interpreter will tell you what type something belongs to
    - >>> type("x+y")

      <class 'str'>
    - >>> type(3)

      <class 'int'>
    - >>> x = 3
    - >>> type(x)

      <class 'int'>
- Variables are names that refer to instances of a value

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Objects

- *Objects* are Python's abstraction for data
  - All data in a Python program is represented by objects or by relations between objects
- Every object has an identity, a type, and a value
  - An object's *identity* never changes – can be thought of as it's memory address – id()
  - An object's *type* is also unchangeable – determines operations the object supports and defines the possible values for objects of that type
  - An object's *value* may be able to change – determined by the *mutability* of the object

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ID example

```
>>> x = 10
>>> id(x)
2028393664
>>> id(10)
2028393664
>>> y = 10
>>> id(y)
2028393664
>>> x = 5
>>> id(x)
2028393584
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Objects & Mutability

- An object whose value can change is *mutable*
- An object whose value is unchangeable once it is created is *immutable*
  - The value of an immutable container object that contains a reference to a mutable object can change when the latter's value is changed
  - The container is still considered immutable, because the collection of objects it contains cannot be changed
- An object's mutability is determined by its type
  - Numbers, strings, and tuples are immutable
  - Dictionaries and lists are mutable

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Variable Names

- Variable names:
  - Should be meaningful
  - May be arbitrarily long
  - May only contain letters, numbers, and '_'
  - Must begin with a letter or '_'
  - May contain UPPER- and lower-case letters
    - Variable1 is different from variable1
- Variable names may NOT:
  - Overlap with any Python **keywords**
  - Start with a number
  - Contain anything other than alphanumerics and '_'

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Keywords

- Keywords are:
  - Used by the interpreter to recognize the structure of the program
  - Protected – may not be used as variable names
- Keywords include:
  - 'False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'
- If you're ever unsure, try it in a Python interpreter
  - In practice, you should try to avoid anything Python will pick up as a keyword

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Types

- As mentioned, all objects in Python have a type. The built-in types available to all Python programs are:
  - Common:
    - bool, bytes, dict, int, list, set, str, tuple
  - Rare:
    - bytearray, complex, float, frozenset, memoryview, object
- Notes:
  - Objects are used everywhere, but the object built-in is rarely used directly.
  - Built-ins are viewable by running: dir(__builtins__) or viewing the documentation.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Types: Numeric

- Three distinct numeric types
  - Plain integers (<class 'int'>)
    - Booleans are a subtype of plain integers
    - RIP <type 'long'>
  - Floating point numbers (<class 'float'>)
  - Complex numbers (<class 'complex'>)
- Numbers are created by numeric literals or as the result of built-in functions and operators

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Types: Numeric

- By default, integer division yields a floating point number

```
>>> 10 / 3
3.3333333333333335
```

```
>>> 10 / 2
5.0
```

- Can force Python 2.x style integer division

```
>>> 10 // 3
3
```

- Numeric literals containing a decimal point or an exponent yield floating point numbers

```
>>> 10.0 / 3
3.3333333333333335
```

```
>>> 10.0 // 3
3.0
```

- Floating point numbers have variable precision based on platform

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Precedence

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence
- Operators with the same precedence are evaluated left to right
- For mathematical operators, Python follows mathematical convention (PEMDAS)
    - <u>P</u>arentheses
    - <u>E</u>xponentiation
    - <u>M</u>ultiplication & <u>D</u>ivision (same precedence)
    - <u>A</u>ddition & <u>S</u>ubtraction (same precedence)
- Non-mathematical operators usually evaluated FIRST

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Sequence

- Python sequences represent finite ordered sets indexed by integers
- Mutability is an important characteristic of sequence types
- Mutable sequences can be changed after creation
  - list
  - bytearray (binary)
- Immutable sequences may NOT be changed after creation
  - bytes (binary)
  - range
  - str (text)
  - tuple
- An iterable is an object capable of returning its members one by one
  - Sequences are a very common type of iterable

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Sequence Operations

- All sequence objects have a common set of methods
- Some sequence types have additional member methods

OPERATION	RESULT
x in s	True if an element of x is in s, else False
x not in s	False if an element of x is in s, else True
s + t	the concatenation of s and t
s * n, n * s	n shallow copies of s concatenated
s[i]	i'th element of s, origin 0
s[i:j]	slice of s from i to j
s[i:j:k]	slice of s from i to j with step k
len(s)	length of s
min(s)	smallest element of s
max(s)	largest element of s
s.index(x[, i[, j]])	index of the first occurrence of x in s (at or after index *i* and before index *j*)
s.count(x)	total number of occurrences of x in s

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Indexing and Slicing

- Indexing
  - Retrieves items from an offset
  - Negative indexing allowed – adds length of object to offset
    - x[-2] is the second to last item – x[len(x)-2]
  - x[0] is the first item (zero based)
- Slicing
  - Retrieves sections of an item from one offset to another
  - x[1:n] returns from the second item (offset 1) up to, but not including, the nth item
  - x[1:] returns from the second item (offset 1) to the end
  - x[:-1] returns from the first item (offset 0) up to, but not including, the last item (len(x)-1)

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: String

- A string is an IMMUTABLE sequence of characters – may not be modified in place – there is no character type
- Strings are created by assignment

```
>>> str_x = "asdf"
```

- Strings may be accessed one character at a time by using the [] (bracket) operator and an index
  - Indexes may be any expression or variable
  - Indexes must evaluate to integers or you will get an exception

```
>>> str_x[0]
'a'
>>> str_x['a']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: string indices must be integers
>>>
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: String Operations

```
ystring = "The worst of times"                              # assignment
xstring = "The best" + " of times"                          # concatenation
xstring = xstring * 3                                       # repetition
xstring[0]                                                  # indexing
xstring[-1]                                                 # indexing from
end
xstring[1:4]                                                # slicing
len(xstring)                                                # length method
if xstring < ystring:                                       # comparison
if "rst" in ystring:                                        # inclusion
xstring = xstring + "\r\nnewlines"                          # escaped control
xstring += 'single quotes' + """triple quotes""" + r"\nraw"  # raw
```

- upper(), lower(), replace(), split()
- Plenty of other functions as well. Do a dir("") or check the Python module docs

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Slicing: Examples

```
>>> x = "abcdefg"
>>> x[0]
'a'
>>> x[-1]
'g'
>>> x[1:2]
'b'
>>> x[2:]
'cdefg'
>>> x[:2]
'ab'
>>> x[-4:5]
'de'
>>> y = ["abc", "def", "ghi"]
>>> y[1]
'def'
>>> y[2][0]
'g'
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Formatting

- Python supports formatting values into format strings.
- Format strings contain "replacement fields" surrounded by curly braces {}. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output.
- Fields can be either a number or a named keyword
- Numbers refer to positional arguments, and keywords refer to the named keyword
- If numerical arguments are in order (0, 1, 2), they can be omitted

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Formatting *(continued)*

ACTP ADVANCED CYBER TRAINING PROGRAM

- Format string formatting examples:

```
"First, thou shalt count to {0}"  # References first positional argument
"Bring me a {}"                   # Implicitly references the first positional
                                   # argument
"From {} to {}"                   # Same as "From {0} to {1}"
"My quest is {name}"              # References keyword argument 'name'
"Weight in tons {0.weight}"       # 'weight' attribute of first positional arg
"Units destroyed: {players[0]}"   # First element of keyword argument
                                   # 'players'.
```

ManTech

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Formatting *(continued)*

- Format string example script:

```
my_name = input('What is your name? ')
print('Hello, {}!'.format(my_name))
my_age = input('Hey {0}, how old are you? '.format(my_name))
print("Wow, {}! You're {}!?".format(my_name, my_age))
print('How much {0} could a {0}{1} {1}?'.format(
         'wood', my_name.lower()
))
print('Goodbye, {name}!'.format(name=my_name))
```

- Format string example execution:

```
python formatting.py

What is your name? Chuck
Hello, Chuck!
Hey Chuck, how old are you? 37
Wow, Chuck! You're 37!?
How much wood could a woodchuck chuck?
Goodbye, Chuck!
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Formatting *(continued)*

- "Format specifications" are used within replacement fields contained within a format string to define how individual values are presented. Refer to the section in the docs entitled "Format Specification Mini-Language" for more information.

- Example:

```
>>> age = 37
>>> '{: 4}'.format(age)   # format the value into a 4-length string
'  37'
>>> '{:^4}'.format(age)   # center-align the value into a 4-length string
' 37 '
>>> '{:04}'.format(age)   # print with extra spaces filled with zeroes
'0037'
>>> '{:04x}'.format(age)   # print the hexadecimal form
'0025'
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Formatting, alternate

- An older form of string formatting is the C-style sprintf-format (used mostly prior to python 2.7):

```
>>> "Wow %s! You're %d!?" % (my_name, my_age)
"Wow Chuck! You're 37!?"
```

- A newer form of string formatting uses "f-strings" (python 3.6+):

```
>>> f"Wow {my_name}! You're {my_age}!?"
"Wow Chuck! You're 37!?"
```

- Depending upon the environment you're working in, you may be required to use an older formatting style, but most places use at least python 2.7, which supports plain old format().

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Concatenation & Repetition

- 'x + y' makes a new sequence object with the contents of both operands (x and y)
- 'x * n' makes a new sequence object with n copies of the sequence operand (x)

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Method Examples

```
>>> #Concatenation
>>> x = "asdf"
>>> y = x + " space " + "words"
>>> print(y)
asdf space words
>>>
>>> #Repetition
>>> z = x * 3
>>> print(z)
asdfasdfasdf
>>>
>>> #Exploding
>>> my_list = list(x)
>>> print(my_list)
['a', 's', 'd', 'f']
>>>
>>> #Imploding
>>> new_x = ''.join(my_list)
>>> print(new_x)
asdf
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Indexing and Slicing

```
>>> x = "so get this"
>>> x.split("get")
['so ', ' this']
>>> x.replace("get", "forget")
'so forget this'
>>> where = x.find("get th")
>>> x[where] = "what is "
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> my_list = len("get th")
>>> x = x[:where] + "what " + x[where + 1:]
>>> x
'so what is'
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

String Mini-Lab

- Given the string "The quick brown fox jumped over the lazy dog":
  - Replace "the" with "a"
  - Determine the length of the string
  - Make it uppercase
  - Programmatically determine if the word "Brown" is in the string
  - Programmatically determine how many words are in the sentence
    - Assume delimited by space

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Bytes

- A bytes object is an IMMUTABLE built-in type for handling binary data.
- Bytes may be accessed individually by using the square bracket operator [], like strings.
- If the slice given is simply an index, an integer is returned. Otherwise, a bytes object is returned.
- While bytes are most frequently acquired via reading from a file or socket, a binary literal can be included in your snippets by prefixing a string literal with a letter 'b':
  - Single quotes: b'still allows embedded "double" quotes'
  - Double quotes: b"still allows embedded 'single' quotes".
  - Triple quoted: b'''3 single quotes''', b"""3 double quotes"""

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Bytes *(continued)*

- Typically, binary literals are used as constants.

```
>>> EURO_CURRENCY_BYTES = b'\xe2\x82\xac'
>>> type(EURO_CURRENCY_BYTES)
<class 'bytes'>
```

- Bytes can be decoded into strings, and strings can be encoded into bytes. The default encoding in calls to encode() or decode() is UTF-8.

```
>>> EURO_CURRENCY_STRING = EURO_CURRENCY_BYTES.decode()
>>> type(EURO_CURRENCY_STRING)
<class 'str'>
>>> my_string = 'I found a $5 note!'
>>> print(my_string.replace('$', EURO_CURRENCY_STRING))
'I found a €5 note!'
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Bytes vs. Strings

- Converting to a string from bytes is easy!
- For a string, you will use the .encode() method
- For a bytes object, you will use .decode()
- This allows easy translation between the two data types.
- Strings in Python 3 are capable of handling Unicode

```
>>> beta = "\u03b2"
>>> bytes_beta = beta.encode('utf-8')
>>> print(bytes_beta)
b'\xce\xb2'
>>> print(bytes_beta.decode('utf-8'))
β
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Types: List

- A list is a MUTABLE sequence of values sometimes called elements or items
- Lists may be thought of as arrays of object references
- Lists are created through assignment or the list constructor

```
>>> list_x = []
>>> list_y = list()
```

- List objects support:
  - Additional operations that allow in-place modification of the object
  - Access by offset
  - Variable length and makeup and are arbitrarily nestable

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common List Operations

OPERATION	RESULT
s[i] = x	item *i* of s is replaced by x
s[i:j] = t	slice of s from *i* to *j* is replaced by the contents of the iterable *t*
del s[i:j]	same as s[i:j] = []
s[i:j:k] = t	the elements of s[i:j:k] are replaced by those of *t*
del s[i:j:k]	removes the elements of s[i:j:k] from the list
s.append(x)	appends x to the end of the sequence (same as s[len(s):len(s)] = [x])
s.clear()	removes all items from s (same as del s[:])
s.copy()	creates a shallow copy of s (same as s[:])
s.extend(t) or s += t	extends s with the contents of *t*
s *= n	updates s with its contents repeated *n* times
s.insert(i, x)	inserts x into s at the index given by *I* (same as s[i:i] = [x])
s.pop([i])	retrieves the item at *i* and also removes it from s
s.remove(x)	remove the first item from s where s[i] is equal to x
s.reverse()	reverses the items of s in place

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

List Operation Examples

```
>>> list_x = []                     # create an empty list
>>> list_y = [1, 2, 3]              # create a non-empty list
>>> list_y
[1, 2, 3]
>>> list_x.append("asdf")           # add (to the end) "asdf"
>>> list_x
['asdf']
>>> list_x.insert(0, ";lkj")        # insert at index 0 ";lkj"
>>> list_x
[';lkj', 'asdf']
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

List Operation Examples *(continued)*

```
>>> list_x
[';lkj', 'asdf']
>>> list_y
[1, 2, 3]
>>> list_x.append(list_y)  # create a nested list
>>> list_x
[';lkj', 'asdf', [1, 2, 3]]
>>> list_z = [4, 5, 6]     # create a new list
>>> list_y
[1, 2, 3]
>>>
>>> # add to the end of list_y all elements of list_z
>>> list_y.extend(list_z)
>>> list_y
[1, 2, 3, 4, 5, 6]
>>> list_x
[';lkj', 'asdf', [1, 2, 3, 4, 5, 6]]
>>>
```

- Why does list_x change when we extend list_y?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Sequence Iteration

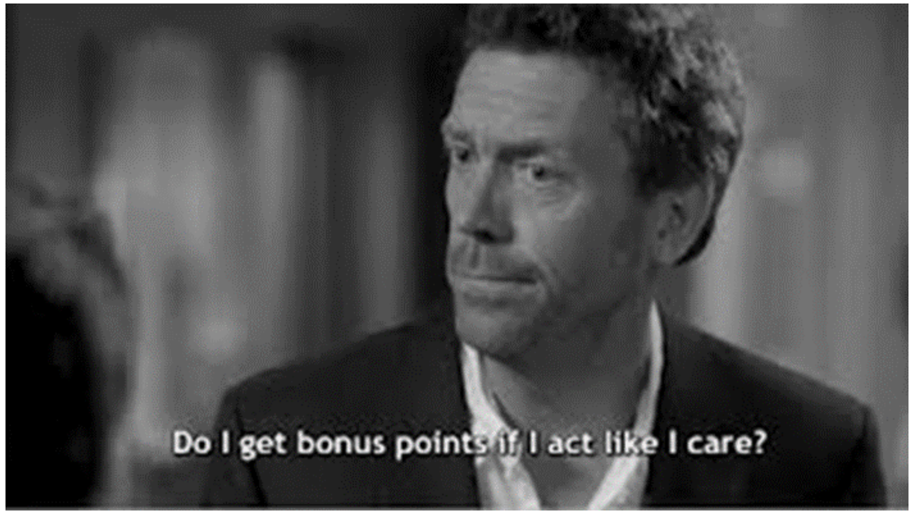
- Two common methods of list iteration depending on the goal
  - By Index
  - By Element

```
>>> import string
>>> lower_list = list(string.ascii_lowercase)
>>> lower_list
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
'z']
>>> # By Index
>>>  for index in range(len(lower_list)):
 . . .       print("{} @ {}".format(lower_list[index], index))
>>>  # By Element
>>>  for element in lower_list:
 . . .       print("{} @ {}".format(element, lower_list.index(element)))
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Mini-Lab!

- Take the string "The quick brown fox jumps over the lazy dog" from the previous mini-lab and reverse it.
  - Do not reverse the words themselves, just their location in the string
    - i.e. Your output should look like “dog lazy the over jumps fox brown quick The”
  - For fake bonus points, see how many ways you can reverse the string!



Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Bytearray

- A bytearray object is a MUTABLE built-in that is used to hold and operate on binary data.
- The bytearray object has many functions that make it behave like both a list and bytes.
- The constructor can be called with a bytes object or an integer, to specify the length. This allows the bytearray object to behave like a traditional buffer.

```
>>> ba = bytearray(10)
>>> ba
bytearray(b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
>>> ba = bytearray(b'hello!')
>>> ba
bytearray(b'hello!')
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Bytearray

- The bytearray object is useful when low-level access to bytes is required and regular. Here is a simple example:

```
>>> ba = bytearray(my_string.encode())
>>> ba
bytearray(b'I found a $5 note!')
>>> ba.index(b'!')
17
>>> ord('?')
63
>>> ba[17] = 63
>>> ba
bytearray(b'I found a $5 note?')
>>> my_new_bytes = bytes(ba)
>>> my_new_bytes
b'I found a $5 note?'
```

- Question: Why did this example not show replacing the '$' with the euro note from earlier?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Bytearray

- Bytearray quirkiness!

```
>>> ba = bytearray(my_string.encode())
>>> ba
bytearray(b'I found a $5 note!')
>>> ba.index(b'$')
10
>>> id(ba)
17100576
>>> ba[10] = EURO_CURRENCY_BYTES
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'bytes' object cannot be interpreted as an integer
>>> ba[10:11] = EURO_CURRENCY_BYTES
>>> ba
bytearray(b'I found a \xe2\x82\xac5 note!')
>>> id(ba)
17100576
```

- Note: notice the location of the bytearray has not changed.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Bytearray

- Here is an example that operates on each individual byte in a bytearray:

```
>>> my_bytearray
bytearray(b'I found a $5 note!')
>>> for index in range(len(my_bytearray)):
...     my_bytearray[index] = my_bytearray[index] ^ 1
...
>>> my_bytearray
bytearray(b'H!gntoe!'!%4!onud ')
>>> for index in range(len(my_bytearray)):
...     my_bytearray[index] = my_bytearray[index] ^ 1
...
>>> my_bytearray
bytearray(b'I found a $5 note!')
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Tuple

- A tuple is an IMMUTABLE sequence - cannot be modified once created
  - Can contain any type
  - Are indexed by integers and can be accessed by offset
  - Support slicing operations
- Tuples are created by:
  - Assignment
  - Using the *tuple()* constructor

```
>>> tuple_x = ()
>>> tuple_y = tuple()
```

  - A tuple with a single item must be constructed with a trailing ","

```
>>> tuple_z = "a",
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common Tuple Operations

- Same as built-in sequence methods!

OPERATION	RESULT
x in s	True if an element of x is in s, else False
x not in s	False if an element of x is in s, else True
s + t	the concatenation of s and t
s * n, n * s	n shallow copies of s concatenated
s[i]	i'th element of s, origin 0
s[i:j]	slice of s from i to j
s[i:j:k]	slice of s from i to j with step k
len(s)	length of s
min(s)	smallest element of s
max(s)	largest element of s

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Tuple Operation Examples

```
>>> tuple_x = "a","b","c","d"       # Create a basic tuple
>>> tuple_x
('a', 'b', 'c', 'd')
>>> for char in tuple_x:            # Iterate through the items
...     print(char, " ", end="")    # and print them on the same line
...
a b c d
>>> "".join(tuple_x)                # Join the items into a string
'abcd'
>>> char_a = "a"
>>> if char_a in tuple_x:           # Check for membership "in" tuple_x
...     print("in there")
...
in there
>>> tuple_x[0:3]                    # Slicing
('a', 'b', 'c')
>>> tuple_x * 3                     # Repetition
('a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd')
>>> "".join(tuple_x * 3)            # Repetition and string joining
'abcdabcdabcd'
>>> x , y = tuple_x[0:2]            # Only grab the first 2 items
>>> print("x = {0}; y = {1}".format(x, y))
x = a; y = b
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: Dictionary

- Dictionaries are MUTABLE objects which describe a one to one relationship (map) between a set of key : value pairs
  - Keys can be any hashable type and are like list indices
  - Values can be any Python object
  - If a key is used more than once, the LAST used value will be stored
  - Dictionaries are stored in insertion order
  - Dictionaries may NOT be sliced
- Dictionaries are created by:
  - Placing comma separated list of key : value pairs within {} (braces)
  - Using the dict() constructor

```
>>> dict_x = {"zero":"a", "one":"s", "two":"d", "three":"f"}
>>> dict_y = dict(zero="a", one="s", two="d", three="f")
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common Dictionary Operations

OPERATION	RESULT
d[i] = x	item i of d is replaced by x
d.clear()	remove all items from d
d.copy()	a shallow copy of d
d.fromkeys(S[,v])	new dict with keys from S and values equal to v; v defaults to None
d.get(k[,v])	d[k] if k in d, else v. v defaults to None.
k in d	True if d has a key k, else False
d.items()	view of d's (key, value) pairs, as 2-tuples
d.keys()	view of d's keys
d.pop(k[,v])	remove the specified key and return the corresponding Value; if key k not found , v is returned if given otherwise Key Error is raised
d.popitem()	remove and return some (key, value) pair as a 2-tuple; but raise KeyError if d is empty – removes last item added from dictionary
d.setdefault(k[,v])	d.get(k,v), also set d[k]=v if k not in d
d.update(E, **F)	add all elements of E to d replacing values of s with values of E whose keys overlap; same with extended list of dicts **F
d.values()	view of d's values

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Dictionary View Objects

- Objects returned by dict.keys(), dict.values(), and dict.items()
- Provide dynamic view of the dictionary's entries
- Can be iterated over to yield the respective data

```
>>> d = {'one': 1, 'two': 2, 'three': 3}
>>> keys = d.keys()
>>> values = d.values()
>>> for k, v in d.items():
        print(k, v)
>>> print('keys: {}'.format(list(keys)))
>>> print('values: {}'.format(list(values)))
>>> del d['one']
>>> print(list(keys))
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Dictionary Creation Operations

```
>>> dict_x = {}             # Create an empty dictionary
>>> dict_x["one"] = 1       # Add an item to a dictionary
>>> dict_x = {"one" : 1}    # Create a dictionary with an item
>>> dict_y["two"] = 2       # Attempt to add an item to an undeclared dictionary
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'dict_y' is not defined
>>>
>>> dict_y = {}             # Create another empty dictionary
>>> dict_y["two"] = 2       # Set item key "two" equal to 2
>>> dict_y["two"] = 3       # Set item key "two" equal to 3
>>> dict_y["three"] = 3     # Add a new item "three" = 3
>>> dict_y
{'two': 3, 'three': 3}
>>> dict_y["one"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'one'
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Dictionary Access Examples

```
>>> dict_y
{'two': 3, 'three' : 3}
>>> dict_y.items()           # Get all items - returns a list of tuples
dict_items([('two', 3), ('three', 3)])
>>> dict_y.keys()            # Get all keys - returns a list
dict_keys(['two', 'three'])
>>> dict_y.values()          # Get all values - returns a list
dict_values([3, 3])
>>> dict_y.popitem()         # Remove and return an arbitrary item
('two', 3)
>>> dict_y
{'three': 3}
>>> dict_y.pop("three" , 0) # Remove and return the value associated with key
3
>>> dict_y
{}
>>> dict_y.pop("three", 0) # Remove and return the value associated
0                           # with the key - default to 0 if key not found
>>> dict_y
{}
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Dictionary Access Examples *(continued)*

```
>>> dict_x
{'one': 1}
>>> dict_y.clear()         # Clear all items from a dictionary
>>> dict_y["four"] = 4     # Add an item to a dictionary
>>> dict_y.update(dict_x) # Add all unique items from dict_x to dict_y
>>> # Overwrite all items in dict_y who share a key in dict_x
>>>
>>> dict_y
{'four': 4, 'one': 1}
>>> dict_x
{'one': 1}
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Built-in Type: File

- Python's built-in file type is an abstraction of the standard C FILE* library and provides methods for accessing data contained in files
  - Files can be accessed in read, write, or append mode (r, w, a)
  - Files can be accessed as bytes data or "special Python text"
  - If opened for write or append access, a file will be created if it does not exist
  - If a file operation fails for an I/O related reason an IOError exception will be raised
- Files may be accessed with the open command

```
>>> fh = open("example.txt", "wb")
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common File Operations

OPERATION	RESULT
f.close()	Close handle to file f
f.closed	True or False. True if f is closed
f.fileno()	integer "file descriptor".
f.flush()	Flush the internal I/O buffer.
f.isatty()	True or False. True if the file is connected to a tty device.
f.mode	file mode ('r', 'U', 'w', 'a', possibly with 'b' or '+' added)
f.name	File name
f.read([size])	read at most size bytes, returned as a string.
f.readline([size])	next line from the file, as a string. Retain newline. A non-negative size argument limits the maximum number of bytes to return
f.readlines([size])	list of strings, each a line from the file. Call readline() repeatedly and return a list of the lines so read. The optional size argument, if given, is an approximate bound on the total number of bytes in the lines returned.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common File Operations *(continued)*

OPERATION	RESULT
f.seek(offset)	Move to new file position
f.tell()	current file position, an integer
f.truncate([size])	Truncate the file to at most size bytes. Size defaults to the current position, as returned by tell
f.write(str)	Write string str to file.
f.writelines(seq_of_strings)	Write the strings to the file.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common File Operations *(continued)*

```
>>> fh = open("example.txt","rb")  # Open file "example.txt" for read, bytes
>>> if not fh.closed:              # Determine if fh is closed and if not…
...     fh.fileno()                # ask for the fileno
...
3
>>> data = fh.read()               # Returns entire contents of fh as a
string
>>> fh.seek(0)                     # Set file index to 0
>>> fh.readline()                  # Reads a line from the file
b'line 0\r\n'
>>> fh.readline(5)                 # Read at most 5 bytes from the current
b'line '                           # position in the file
>>> fh.readline()                  # Read to the next line – returns the
b'1\r\n'                           # 'rest' of the line from the last
readline
>>> fh.mode
'rb'
>>> fh.name
'example.txt'
>>> fh.seek(0)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common File Operations *(continued)*

```
>>> fh.seek(0)                                # Seek to the 0th position
>>> for count in range(0, 100):               # iterate from 0 to 99
...     line = fh.readline()                  # get the next line and
...     print(line)                           # print it
...
b'line 0\r\n'
b'line 1\r\n'
…
b'line 99\r\n'
>>>
>>> fh.tell()                                 # Tell us our offset
890
>>> fh.seek(8*25)                             # Seek to the 200th position
>>> fh.truncate()                             # and TRUNCATE!!!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
io.UnsupportedOperation: File not open for writing
>>> fh.close()
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common File Operations *(continued)*

- Introducing the with construct

```
>>> with open("ex.txt", "rb") as fh:
...     for line in fh:
...         print(line)
```

```
>>> fh = open("ex.txt", "rb")
>>> try:
...     for line in fh:
...         print(line)
... finally:
...     fh.close()
```

```
>>> with open("example.txt", "ab") as fh:   # Open example.txt for APPEND!!!
...     fh.seek(8*25)
...     fh.truncate()
>>> fh.closed
True
>>> with open("example.txt", "rb") as fh:   # Open example.txt for READ
...     data = fh.read()
...     fh.tell()
200
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common File Operations *(continued)*

- Python tries to "help" you when you open without the binary flag...

```
>>> with open("example.txt", "rb") as fh:
...     data = fh.read()
...
>>> data[:40]
b'line 0\r\nline 1\r\nline 2\r\nline 3\r\nline 4\r\n'
>>>
>>> with open("example.txt", "r") as fh:
...     data = fh.read()
...
>>> data[:35]
'line 0\nline 1\nline 2\nline 3\nline 4\n'
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Common File Operations *(continued)*

- Bytes are required for a file opened with the binary flag

```
>>> fh = open("foo.txt", "wb")
>>> fh.write("Python is the best!\n")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: a bytes-like object is required, not 'str'
>>> fh.write(b"Python is the best!\n")
20
>>> fh.write("Python is the best!\n".encode())
20
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

"Bit-string" Operations

- Integers support additional operations that make sense only for bit-strings
  - Negative numbers are treated as their 2's complement value
  - Priorities of the binary bitwise operations are all lower than the numeric operations and higher than the comparisons
  - ~ has the same priority as the other unary numeric operations (+ and -)

OPERATION	RESULT
x | y	bitwise or of x and y
x ^ y	bitwise exclusive or of x and y
x & y	bitwise and of x and y
x << n	x shifted left by n bits
x >> n	x shifted right by n bits
~x	the bits of x inverted

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

"Bit-string" Operations, Continued

```
>>> # these are all hex numbers - NOT BINARY
>>> style = "0x{:08x}"
>>> style.format(0x00000010 | 0x00000001)   # or
'0x00000011'
>>> style.format(0x00000010 & 0x00000001)   # and
'0x00000000'
>>> style.format(0x00000010 ^ 0x00000001)   # xor
'0x00000011'
>>> style.format(0x00000010 ^ 0x00000000)   # xor
'0x00000010'
>>> style.format(0x00000001 << 4)           # shift left 4
'0x00000010'
>>> style.format(0x00000001 << 8)           # shift left 8
'0x00000100'
>>> style.format(0x00000001 << 12)          # shift left 12
'0x00001000'
>>> style.format(0x00000001 << 16)          # shift left 16
'0x00010000'
>>> style.format(0x00010000 >> 12)          # shift right 12
'0x00000010'
>>> def bin(x):
...     return "".join(x & (1 << i) and "1" or "0" for i in range(7,-1,-1))
...
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Lab 3

Lions, Tigers & Built-i

Oh My!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- Write a Python script that will:
- Insert the numbers 0-25 into a list
- Create a string that contains the lowercase letters of the alphabet
- Create a dictionary where the keys are the letters of the alphabet and the values are the corresponding position (a=0, b=1, etc.)
- Now, open a file named "mypairs.txt" and write the corresponding keys – value pairs to the file.
  - Try writing the key-value pairs to the file both in alphabetical and reverse alphabetical order

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Type Hinting

- Type hints are performed using Python annotations. They are used to associate types with variables, parameters, function arguments as well as their return values, class attributes, and methods.
  - **Adding type hints currently has no runtime effect: these are only hints and are not enforced.**
  - You can perform type hints with any of the following basic built-in types:
    - int, float, str, bool, bytes, list, tuple, dict, set, frozenset, None

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Type Hinting *(continued)*

- Variable annotations follow this syntax:
  - *<variable_name>: <variable_type> = <variable_value>*
  - Examples:
    - a: int = 3
    - b: float = 2.4
    - c: bool = True
    - d: list = ["A", "B", "C"]
    - e: dict = {"x": "y"}
    - f: set = {"a", "b", "c"}
    - g: tuple = ("name", "age", "job")

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Type Hinting *(continued)*

- Function annotations
  - Type hinting the returned value is done using an arrow ->
    - def add_numbers(x: type_x, y: type_y, z: type_z= 100) -> return_type:

      ```
          return x + y + z
      ```
    - → If a function doesn't return anything, you can set the return type (after the arrow) to None

- Class Annotations
  - You can even annotate attributes and methods inside your classes.
  - class Person:

    ```
        first_name: str = "John"
        last_name: str = "Does"
        age: int = 31
    ```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Type Hinting - Advantages

- Using type hints, especially in functions, document your code by providing an informative signature that clarifies the assumptions about the types of arguments and the type of the result produced
- Informing the types removes a cognitive overhead and makes the code easier to read and debug. With the types of inputs and outputs in mind, you can easily reason about your objects and how they can be coupled
- Type hints improve your code editing experience

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Values are the basic building blocks of within a program
- All values have an underlying type
- Everything in Python is an object
- Objects whose value can change are MUTABLE
- Objects whose values CANNOT change are IMMUTABLE
- Use meaningful variable names
- Keywords are used by the interpreter to recognize structure in the program
- 3 numeric types: int, float, complex
- Python uses the standard mathematical precedence convention

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review *(continued)*

- Sequences are finite ordered sets indexed by integer values
- Sequences include list, range, str, tuple
- Indexing retrieves items from an offset in a sequence
- Slicing retrieves contiguous groups of items from a sequence
- Python supports C sprintf style syntax for formatted output
- Integers support operations that make sense only for bit-strings
- Use type hinting to help document code and improve the code editing experience
- Questions?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Statements

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Given a workstation and technical documentation, the student will be able to use basic statements in Python.
- Enabling Objectives
  - Describe Python program structure
  - Describe general Python syntax
  - Describe the syntax of assignment statements, expressions, print statements, if statements, and iteration
  - Use assignment statements, expressions, print statements, if statements, and iteration
  - Write a loop

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

Program Structure

- Major components:
  - imports
  - classes / functions
  - top-level code
- Blocks defined by whitespace
- Always use spaces, never tabs
- Notice the __main__, what does that do?

```
import module


def foo(arg):
    x = 5 + arg
    return x


if __name__ == "__main__":
   print(foo(5))
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Program Structure *(continued)*

```
def foo(arg1,                       # parenthesis throw blocks out the window
        arg2):                      # use it for long if statements, etc.


    d = {0 : "a",                   # works for dictionaries, too.
         1 : "b"}                   # most built-ins behave the same.


    arg1 = d[0]; arg2 = d[1]        # semicolons delimit statements; Do NOT use!


    return arg1 + arg2              # what does this return?
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Assignment

```
x = [0, 1]              # x is two element list
x = (0, 1)              # x is two element tuple
x = 0, 1                # x is still a two-element tuple
x, y = 0                # TypeError: cannot unpack non-iterable int
x, y = 0, 1             # x = 0, y = 1 "Tuple unpacking/packing"
x, y = 0, (1, 2)        # x = 0, y = (1, 2)
x, y = 0, 1, 2          # ValueError: too many values to unpack
x += 1                  # shorthand for x = x + 1.  (No ++ or --!)
x *= 2                  # shorthand for x = x * 2.
x -= 2                  # shorthand for x = x - 2.
x **= 2                 # shorthand for x = x^2. (exponents).
x /= 2                  # shorthand for x = x / 2.
x //= 2                 # shorthand for x = x // 2.(floored
division)
x %= 2                  # shorthand for x = x % 2.
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Expressions

- An expression is a combination of values, variables, and operators
  - Lone variables and values are considered expressions*
  - In the interpreter, the result of an expression is printed
  - In a script, lone expressions are orphans and do nothing

```
>>> x = 3
>>> 10 + 3
13
>>> 10
10
>>> 3
3
>>> x + 10
13
```

```
python expression.py

# expression.py
x = 3
10 + 3
10
3
x + 10
```

* If the variable has already been assigned

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Expressions, continued

• Math operators:	+, -, *, /, **, %
• Shift operators:	>>, <<
• Bitwise operators:	&, |, ^, ~
• Comparison operators:	>, <, >=, <=, ==, !=
• Assignment operators:	=, +=, -=, *=, /=, **=, %=
• Logical operators:	and, or, not
• Other operators:	in, del, with

Python expressions are like those in C

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python's print

- print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
  - Calls __repr__ or __str__ (if present) on the expression result to format for output. If not present, a generic '<object scope.name at address>' is displayed
  - Can assign keyword values to manipulate output of code.

```
>>> print(1+2)                      # 3\n
>>> print(1, 2)                     # 1 2\n
>>> print(1 + "a")                  # exception: unsupported operand…
>>> print(1, "a")                   # 1 a\n
>>> print(1, "a", end="")           # 1 a # no trailing CRLF
>>> print("o" "n" "e") # one\n #parser concatenation
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python's if

- If/else/elif. No parens required around expressions. Colon required at expression end
- Empty strings, lists, tuples, dictionaries, etc. all evaluate to False
- Populated strings, lists, tuples, dictionaries, etc. evaluate to True
- None evaluates to False
- 0 evaluates to False
- Boolean operators are 'and', 'or', and 'not'
- Short-circuits from highest to lowest

```
if expression:
    statement
    statement
elif expression:
    statement
else:
    statement
    statement
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python's if *(continued)*

```
x = 1
y = 2
if x == 1 and y == 2 or x == 3:
   print("woot")

if not x:
   print("wootwoot")

x = (1, 2, "a")
if "a" in x:
   print("Python rocks.")
else:
   print("Way better than Perl or Java.")
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python Iteration

- for val in sequence: - lists, tuples, or dictionary views
- for val in iterator: - See next slide
- while expr: - Loops while expr evals to True
- 'break' moves execution to first instruction after loop block
- 'continue' moves execution to top of loop block

```
>>> for x in range(3):
...     print(x)
...
0
1
2
```

```
>>> for x in (1, 2, 3):
...     print(x)
...
1
2
3
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Sequence vs Iterator

- Sequences – pre-populated linear set
- Iterators – code which produces next element at evaluation

```
>>> for x in [1,2,3,4,5]:
...     print(x)
```

```
>>> for x in iter([1,2,3,4,5]):
...     print(x)
```

What happens with each? Which is better? Why?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Lab 4

Iteration, Break, Continue & Dictionaries Done Right

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

In a Python script, use the concepts presented to write a loop that:

- Iterates from 1 to 12
- Skips to the next iteration of the loop if the counter is 3
- Exits the loop early if the counter equals 11
- Prints the counter in hex when it is even
- Prints the counter in decimal (not float) when it is odd

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Python programs usually have at least 3 sections: imports, class & function definition(s), and "top level" code
- Expressions are combinations of values, variables, and operators
- In Python, print is a function used to display expressions to stdout
- Branches can be performed with the if statement
- Iteration can be accomplished in many ways in Python – some are a result of the importance of sequence types
- Questions?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ADVANCED CYBER TRAINING PROGRAM

Functions

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objectives
  - Write a basic function
  - Use of generator expressions as well as functions as objects

- Enabling Objectives
  - Describe basic function structure
  - Describe function scope
  - Describe argument matching modes
  - Use function structure as well as scope and argument matching modes to write a basic function
  - Describe functions as objects
  - Describe generator expressions

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Basic Function Structure

- Functions take arguments and optionally return values
  - If no 'return' statement is reached at the end of the function, the return value is None type

```
def square(a):
    return a*a

def wacky(a, b):
    if a < b:
        return '{}[{}]'.format(a, b)
```

```
>>> square(2)
4
>>> square(5)
25
>>> wacky(1,2)
'1[2]'
>>> wacky(2,1)
>>>
>>> wacky(2,1) is None
True
>>> wacky("a", "b")
"a['b']"
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Basic Function Structure *(continued)*

- Functions can return multiple values
  - Returns one value, which is unpacked to match left-hand value(s)

```
def tuple_1(a, b, c):
    return a, b, c

def tuple_2(a, b, c):
    return (a, b, c)

def list_1(a, b, c):
    return [a, b, c]
```

```
>>> tuple_1(1, 2, 3)
(1, 2, 3)
>>> tuple_2(1, 2, 3)
(1, 2, 3)
>>> a, b, c = (1, 2, 3)
>>> (b, c, a)
(2, 3, 1)
>>> a, b, c = list_1(1, 2, 3)
>>> (b, c, a)
(2, 3, 1)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Basic Function Structure *(continued)*

- Functions can return multiple values
  - Unpack must exactly match

```
def tuple_1(a, b, c):
    return a, b, c
```

```
>>> a, b = tuple_1(1, 2, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack
>>> a, b, c, d = tuple_1(1, 2, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Function Scope

- Three levels of scope: local, module/global, and built-in
  - In Python, module scope is called global
  - Lookup: local, module, then built-in
  - You can look at the built-in names with dir(__builtins__)
  - Assigning to variable in local scope leaves globals unmodified

```
# global scope (module-level)
a = 1
b = 2

def f(b):
    # local scope (function level)
    a = 100
    b = 200
    return a, b
```

```
>>> a, b
(1, 2)
>>> f(3)
(100, 200)
>>> a, b
(1, 2)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Function Scope *(continued)*

- Globals are touchy
  - Reading a global works, so long as we don't try to assign to it

```
a = 1
def global_read():
    # a is global: never assigned to
    print(a)

def local_assign():
    # a is local: assigned to before read
    a = 2
    return a*a

def global_write_bad():
    # a is invalid: read from before assign
    a = a + 2
    return a
```

```
>>> a
1
>>> global_read()
1
>>> local_assign()
4
>>> a
1
>>> global_write_bad()
...
UnboundLocalError: ...
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Function Scope *(continued)*

- Globals are touchy
  - We can assign to a global by declaring it global before we use it

```
a = 0
def global_inc_wrong():
    a += 1

def global_inc_right():
    global a
    a += 1
```

```
>>> a
0
>>> global_inc_wrong()
UnboundLocalError:
>>> a
0
>>> global_inc_right()
>>> a
1
>>> global_inc_right()
>>> a
2
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Function Scope *(continued)*

- Globals are touchy
  - We can modify a mutable global without declaring it global
    - From Python's perspective it's a read
  - We can declare multiple variables global in a single statement

```
a, b, c = 0, 1, []

def global_mod(n):
    c.append(n)
    return c

def multiple_globals():
    global a, b
    a, b = b, a + b
    return a, b
```

```
>>> a,b,c
(0, 1, [])
>>> global_mod(1)
[1]
>>> global_mod(2)
[1, 2]
>>> multiple_globals()
(1, 1)
>>> multiple_globals()
(1, 2)
>>> multiple_globals()
(2, 3)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Function Scope *(continued)*

- Local scopes are function level, not indentation level

```
def if_scoping(x):
    if x:
        if x:
            if x:
                a = 1
    return locals()

def loop_scoping():
    column = ['Something'] # overwritten
    for row in range(10):
        for column in range(row*100):
            pass # do something
    return locals()
```

```
>>> if_scoping(False)
{'x': False}
>>> if_scoping(True)
{'a': 1, 'x': True}
>>> loop_scoping()
{'column': 899, 'row': 9}
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Argument Matching Modes

- Default arguments
  - At definition, specified with name=default
  - Non-default arguments not allowed after default arguments

```
def f(a, b=1, c=1):
    return a * b * c
```

```
>>> f(2)
2
>>> f(2, 3)
6
>>> f(2, 3, 5)
30
>>> f(2, b=9)
18
>>> f(a=2, c=4)
8
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Argument Matching Modes *(continued)*

- Positional arguments
  - At call, values separated by commas
  - Can apply a list as positional args with *syntax (Note: * operator here is being used in 2 different ways below)
  - Can capture extra positional args in a list with *listname syntax
    - variable is normally *args by convention
    - args is of type <class 'tuple'>

```
def pos(a, b, c):
    return [a, b, c]

def pos2(a, b, *args):
    return args + (a, b)
```

```
>>> pos(1, 2, 3)
[1, 2, 3]
>>> pos2(*[1, 2, 3])
(3, 1, 2)
>>> pos(*'ABC')
['A', 'B', 'C']
>>> pos2(*range(5))
(2, 3, 4, 0, 1)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Argument Matching Modes *(continued)*

- Keyword arguments
  - At function call, name=value
  - Must follow every positional variable and any *args
  - Can capture or pass extra keywords args with **variable syntax
    - variable is normally **kwargs by convention
    - kwargs is of type <class 'dict'>

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Argument Matching Modes *(continued)*

```
def kw(a, b, **kwargs):
    if kwargs.get('add_ab'):
        kwargs.update(a=a, b=b)
    return kwargs
```

```
>>> kw(1, 2)
{}

>>> kw(1, 2, add_ab=True)
{'a': 1, 'add_ab': True, 'b': 2}

>>> kw(**dict(add_ab=True, b='bbbb', a='aaaa', anotherthing='blah'))
{'a': 'aaaa', 'add_ab': True, 'b': 'bbbb', 'anotherthing': 'blah'}
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

args & kwargs example

- Pass positional variables for *args and keyword variables for **kwargs
- Or pass a *[] for *args or **{} for **kwargs

```
def args_kwargs_test(*args, **kwargs):
    for arg in args:
        print("arg = {0}".format(arg))
    if "foo" in kwargs:
        print("foo's value = 
  {}".format(kwargs["foo"]))


print("Here's one way!")
args_kwargs_test(1,2,3,4,6, foo=5, bar=6)
print("And here's another!")
args_kwargs_test(*[1,2,3,4], **{"foo":5})
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Functions are Objects (everything is!)

- Functions can be passed as variables
  - Applications: callbacks, filters

```
def adder(a, b):
   return a + b

def multer(a, b):
   return a * b

def caller(f, a, b):
   return f(a, b)

>>> caller(adder, 3, 4)
7
>>> caller(multer, 3, 4)
12
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lambdas (Unnamed Functions??)

- You can also create an unnamed function when a function is being used as an object.
- Useful for creating temp functions for sorting / filters

```
def caller(f, a, b):
    return f(a, b)

>>> caller(lambda x, y : x + y, 3, 4)
7
>>> caller(lambda x, y : x * y, 3, 4)
12
>>> f = lambda x : x ** 2
>>> f(5)
25
# Just to blow your mind!
>>> (lambda x, y, z : x + y * z)(4, 5, 6)   # Not recommended!
34
```

ManTech

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Generator Expressions

- Functions can yield instead of returning a value, allowing the function to yield further values
  - When a function returns normally, all local state is lost
  - Yield preserves local state, and returns a generator object which allows the function to iteratively return multiple values
  - Each time __next__() is called on a generator object, the function executes and returns the yielded value, until a yield statement is reached
  - When the function ends, the generator raises StopIteration
  - All of this is handled automatically in the following syntax

```
for var in generator():
   ...
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Generator Expressions *(continued)*

- Example generator: Fibonacci sequence

```
>>> def fib(top):
    a = 0
    b = 1
    while a <= top:
        yield a
        tmp = a + b
        a = b
        b = tmp
```

```
>>> for i in fib(8):
        print(i)

0
1
1
2
3
5
8
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Generator Expressions *(continued)*

- Functions using yield return an object with a __next__() method that returns successive values until function end, then raises StopIteration

```
>>> gen = fib(3)
>>> gen.__next__()
0
>>> next(gen)
1
>>> next(gen)
1
>>> next(gen)
2
>>> next(gen)
3
```

```
>>> next(gen)
Traceback (most recent call
    last):
   File "<pyshell#30>", line 1,
    in <module>
      next(gen)
StopIteration
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ADVANCED CYBER TRAINING PROGRAM

Lab 5

My first def foo:

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- Write a basic function based on the previous section's lab
  - Input
    - A variable that if matched exits the loop
    - The upper and lower boundary for the loops
  - Return
    - The number of even numbers output
- Create a function that modifies a global variable
- Create a function that takes an integer and a variable number of other integers that returns a comma delimited string of all arguments evenly divisible by the initial integer

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Tasks *(continued)*

- Create a function that will MD5 Sum a given file and return the hash value
  - Hint: use the hashlib module and return the hexdigest
- Now create a generator function that has the following inputs and outputs:
  - Input – Path to a directory on disk
  - Output - Returns a generator that yields the filename, hexdigest pairs for every file in that directory
  - Hint: check out listdir in the os module!
  - Bonus: Write a find_dupes function that searches a directory for duplicate files, based on hash value. Print out the hash value and the duplicates
  - Bonus Bonus: Make it work for any number of sub directories (hint: "walk" the directories)
- Create a function that implements ROT13
  - Encoding function where each letter is represented by the letter 13 characters after/before it (wrapping, since there are only 26 letters).
  - A<->N, M<->Z
  - Hint: ord and chr could be helpful here!
  - Bonus: Make it work for upper and lower case!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

ADVANCED CYBER TRAINING PROGRAM

- Functions optionally take arguments and return values – neither is a requirement
- Functions can return one or more values – assignments from returned values must have matched number of variables
- Three levels of function scope – local, module/global and built-in
- Arguments may be matched by default, position, or keyword
- Generators are special functions which yield control back to the caller but retain a state for future iteration
- Questions?

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Classes

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Create and use classes in Python.
- Enabling Objectives
  - Describe why classes are useful
  - Create a class
  - Demonstrate how inheritance is used
  - Describe the difference between classes and modules

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

Why are Classes Useful?

- Encapsulate related behavior and data
- Abstract internal behavior of the class versus how you use the class
- Using inheritance to provide more specific behavior for sub-classes
- Dynamic dispatch allows classes to determine which code to execute for a given call

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Classes

- Python has two class systems – 'old' and 'new'
  - Old classes are the pre-2.2 class system
  - Our focus will be on new classes
- The defining difference between old and new classes:
  - New classes have a chain of base classes leading to 'object'
  - Old classes can have base classes, but do not lead to 'object'
- Everything is an object in Python
  - Attributes – other objects reachable via objectname.attribute
    - These objects can be methods, fields, or even other class instances
  - Class relationships – type and base

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Class Structure

- A class statement defines a new class
  - In the example below, Record is name of the class
  - It has a base (aka parent) class of object
- The __init__ method is a special method - the constructor
- Note the first parameter is self
  - C++ and Java implicitly provide the *this* variable
  - In Python classes you define *self* explicitly in the parameter list
- The assignment in __init__ creates an instance attribute d

```
class Record:
    def __init__(self, d):
        self.d = d
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Class Use (Instances)

- Instantiation looks like a call to the class name
- The type and isinstance builtins inspect the class type
- Attributes are accessed via "." operator
  - This can be chained arbitrarily deep - x.y.foo.bar()

```
>>> rec = Record(12)
>>> type(rec)
<class 'record.Record'>
>>> isinstance(rec, object)
True
>>> isinstance(rec, Record)
True
>>> rec.d
12
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

More on Attributes

- From before: the assignment in __init__ creates an instance attribute
- What is e then?
  - It is a class level attribute
  - It can be accessed from Record.e, from any <instance>.e, or as self.e inside a Record method

```
class Record:
    e = 2
    def __init__(self, d):
        self.d = d
    def __str__(self):
        return self.d
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Methods

- Regular methods are defined with a def statement
  - Indented under the class
  - Remember the initial self parameter
- You don't need to pass self when calling the method
  - Behind the scenes magic: rec.foo(2) => Record.foo(rec, 2)

<appended to previous Record example>

```
    def foo(self, m):
        self.d *= m
        return self.d

>>> rec.foo(2)
24
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Customization Using Special Methods

- Special method names are wrapped in __x__
- __init__ is called for construction
- __new__ and __del__ allow customizing object creation and destruction
  - Be very careful with these two, there are lots of subtle gotchas
- __str__ is called when you print an object or convert via %s
- __repr__ can return a detailed string for the object via %r
  - Call repr(obj) - defaults to __str__ if you don't define your own
- There are **many** of these methods
  - You can use these to emulate container or numeric types
  - Chapter 3.4 of the Python Language Reference has details

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Class Inheritance and Overriding

- This is a 3-level class hierarchy: object->A->B
- The method foo is an override in the subclass B
- The builtin super() is a way to call methods on parent class
  - Less maintainable: calling A.foo(self, d) in B
- super() is useable from any method, not just overrides

```
class A:
    def foo(self, m):
        self.x = m
        return self.x
class B(A):
    def foo(self, m):
        return 4 + super().foo(m)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Mini-Lab!

- Create a Vehicle class with the following attributes:
  - Top speed, number of wheels, color, miles traveled
  - Allow speed, number of wheels, and color to be initialized in the constructor, but miles traveled will always start at 0
  - Create the __str__ member function that will return a string describing your vehicle
  - Create a drive function that will take as input the number of hours to drive. The miles traveled attribute should be updated based on this input and the top speed.
    - No shady mechanics!
  - Create a honk function that prints out the noise made when the horn is honked
- Create a Motorcycle class that is a subclass of your Vehicle class.
  - Same attributes. It should call the Vehicle constructor, always passing in 2 for number of wheels.
  - It should override the honk function (because motorcycle horns sound different, duh)

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Multiple Inheritance

- A class can inherit from multiple parent classes
- Python defines a method resolution order (MRO) for searching parent classes for methods during a call
  - For this simple example, the order is C, A, B
- The MRO can get very complex in messy class hierarchies
- Not as simple as breadth/depth first (C3 linearization)
- http://www.python.org/download/releases/2.3/mro/
- Mixins are the most common use of multiple inheritance

```
class A:
class B:
class C(A, B):
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Multiple Inheritance *(continued)*

```
class A:
    def __init__(self):
        print('In Class A')

class B:
    def __init__(self):
        print('In Class B')

class C(A, B):
    pass

>>> obj = C()
In Class A
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Multiple Inheritance *(continued)*

```
class A:
    def __init__(self):
        print('Entering Class A')
        super().__init__()
        print('Leaving Class A')

class B:
    def __init__(self):
        print('Entering Class B')
        super().__init__()
        print('Leaving Class B')

class C(A, B):
    def __init__(self):
        print('Entering Class C')
        super().__init__()
        print('Leaving Class C')
```

```
>>> obj = C()
Entering Class C
Entering Class A
Entering Class B
Leaving Class B
Leaving Class A
Leaving Class C
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Mixins

- ThreadingMixIn is a minimal class that doesn't stand alone
- By subclassing ThreadingMixIn and HTTPServer you get a modified HTTPServer class that handles threading
- ThreadingMixIn overrides specific methods in HTTPServer to change behavior
- Mixins are usually combinable with multiple other classes

```
class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def doGET():
                pass
class ThreadedHTTPServer(ThreadingMixIn, HTTPServer):
    pass
server = ThreadedHTTPServer(('localhost', 80), RequestHandler)
server.serve_forever()
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Attribute Visibility

- Unlike most other OO languages, there is no real concept of public/protected/private visibility of class attributes
- A single leading _ to a name indicates 'internal use'
  - This applies to modules too! from X import * would ignore _myvar during the import
- You can hide an attribute by prefixing two _ to the name, which mangles the attribute name
- This is just a strong suggestion – it can be bypassed
  - Field defined as __MyField
  - Still visible as _ClassName__MyField

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Properties

- Properties are attributes where usage appears to be field access, but the actual implementation uses methods
  - 'bar.x = 12' will result in a call to setx in the below example

```
class Bar:
    def __init__(self):
        self._x = None
    def getx(self):
        return self._x
    def setx(self, value):
        self._x = value
    def delx(self):
        del self._x

    x = property(getx, setx, delx, "I'm the 'x' property.")
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Decorators

- Any callable Python object used to modify a function/method/class
- "@" denotes use
- Enhance the action of the function/method they decorate
- Common examples include @classmethod, @staticmethod, and @unittest. (skip/skipUnless/skipIf…)

```
@unittest.skipIf(os.path.exists('foo.txt'), 'Because I said so')
def test_bar(self):
    self.assertEqual(5, 5)
```

```
def skipIf(condition,
reason):
    if condition:
        return skip(reason)
    return _id
```

If the file 'foo.txt' exists, skip this test. Otherwise, run it.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Lab 6

Classes

Tasks

- Create a SuperList class that takes in a variable number of ints
- Printing out a SuperList is the same as printing a regular list
- Override __iadd__ such that when you add an int to your SuperList, it adds that int to each element in your SuperList
- Override __setitem__ such that when you set an element in your SuperList, 4x that value gets stored
- Explore some other special methods!!!

```
x = SuperList(10, 11, 12, 13)
x += 5      # [15, 16, 17, 18]
x[1] = 3    # [15, 12, 17, 18]
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Classes encapsulate behavior and data and abstract internal behavior from the use of the class
- Allow for inheritance to provide more specific behavior for subclasses
- A class statement defines a new class
- The __init__ method is a class constructor
- Instantiation looks like a call to the class name – f = foo()
- A class may inherit from multiple parent classes (multiple inheritance)

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Mixins provide minimal classes that do not stand alone, but provide specific behavior
- Python does not really enforce concepts of public/protected/private
- Questions?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Exceptions

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Develop a custom exception class.

- Enabling Objectives
  - Describe a standard Python exception
  - Describe when and when NOT to use exceptions
  - Describe exception catching modes
  - Develop a custom exception class and create a custom child exception class

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Exceptions

- Exceptions are a code flow device for handling exceptional conditions
  - *try* statements begin an exception handling block
  - *raise* statements trigger exceptions
  - *except* statements handle exceptions
- Exceptions are resource intensive and should NOT be used for standard program flow
- There are many built-in exceptions
- When an exception is raised and not dealt with it is called an *unhandled* exception
- When there isn't an exception to do what you want, you can create one!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Exceptions *(continued)*

```
try:
   <statements>
except <name>:
   <statements>
except <name> as <data>:
   <statements>
else: # Executed if no exception is raised
   <statements>

try:
   <statements>
finally:
   <statements>
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Exception Structure (Class Structure)

- User defined exceptions should extend the Exception class
- Exceptions are defined in the exceptions module
  - exceptions never needs to be imported explicitly – it is part of the default namespace

```
class MyError(Exception):
    def __init__(self, msg):
        super().__init__(msg)
       self.msg = msg
    def __str__(self):
        return "Your error was: {}".format(self.msg)
    def __repr__(self):
        return "The repr of your error was: {}".format(self.msg)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Exception Structure (Class Structure)

```
try:
   DoDangerousStuff(None)
except MyError as e:
   print(e)
   print(repr(e))

def DoDangerousStuff(filename):
   if filename is None:
          raise MyError("Dude, you screwed up!")

"Your error was: Dude, you screwed up!"
"The Repr of your error was: Dude, you screwed up!"
```

Any other exceptions raised will not be handled by this statement

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Exception Catching Modes

- Try statements are nestable
- Python selects the first except clause that matches the exception
  - WARNING: an '*except*:' will match **ANY** exception
- Try blocks can contain *except and/or finally*

OPERATION	INTERPRETATION
except:	Catch any exception
except name:	Catch exceptions of type name
except name as value:	Catch exceptions of type name an instance of the exception.
except (name_x, name_y):	Catch exceptions of type name_x and name_y
else:	Run this block only if NO exceptions were caught
finally:	Run this block always

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

When To Use Exceptions

- Exceptions are used in nearly every standard Python library
- Some exceptions can be anticipated
- Some exceptions can result from actual bugs in code
- Commonly encountered exceptions
    - Opening a non-existent file will raise an IOError
    - Accessing a non-existent dictionary key will raise a KeyError
    - Searching a list for a non-existent value will raise a ValueError
    - Calling a non-existent method will raise an AttributeError
    - Referencing a non-existent variable will raise a NameError
    - Mixing data types without coercion will raise a TypeError

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

Lab 7

An exception class
all to myself

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- Starting from your Vehicle mini-lab, develop a VehicleException class.
- Raise it if a negative time to drive is specified.
- Write code to handle that exception in the __main__ block of your script

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Exceptions are a code flow device for handling out of the ordinary conditions
- Exceptions may be built-in or user defined
- Python uses keywords: try, except, finally, else for exception handling
- Python try blocks are nestable
- Questions?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Built-in Tools

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ACTP
ADVANCED CYBER TRAINING PROGRAM

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
    - Demonstrate basic usage of Python built-in tools
- Enabling Objectives
    - Describe how to use various built-in tools
    - Use built-in tools to extend the Python threaded socket server class

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Python pickle

- "Pickling" means turning an object (or group of objects) into a flat byte stream
- Synonyms: Marshalling, serializing, flattening
- Stream representation can be stored to a file or a Python byte string, then reconstructed from either a file or a byte string
- To store to a file:
  - pickle.dump(object, fileobj) -> pickle.load(fileobj)
- To store to a byte string (note the 's' in the method names):
  - pickle.dumps(object) -> pickle.loads(byte string)

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python pickle *(continued)*

- Quick example:

```
>>> class FooClass:
...     pass
>>>
>>> foo = FooClass()
>>> foo.a = 1
>>> foo      # this is our unpickled, normal object
<__main__.FooClass instance at 0x024CA170>
>>> pickle.dumps(foo)     # pickled representation
b"(i__main__\nFooClass\np0\n(dp1\nS'a'\np2\nI1\nsb."
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python pickle *(continued)*

- If we save ('dumps') and then restore ('loads'), we get the original back
- ... and it's like it was never gone!

```
>>> new_foo = pickle.loads(pickle.dumps(foo))
>>> new_foo
<__main__.FooClass instance at 0x02665E90>
>>> new_foo.a
1
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python pickle *(continued)*

- Don't get yourself in a pickle!
- Pickle constraints: some things can't be pickled
    - Locks (from *threading* or other modules)
    - Sockets
    - Nested and Lambda functions
    - Containers that contain unpickleable items (like sockets)
    - etc.
- Trying to pickle these will cause PickleError to be raised

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Pickle Mini-Lab

- Let's try pickling an exception!
- Try it with both dump/load and dumps/loads

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python glob

- Unix style pathname pattern expansion
- Returns list of pathnames that match expression
- Uses os.listdir() under the hood
    - glob can be used recursively
- Can return full path!

```
>>> import glob, os
>>>
>>> os.listdir('c:\testdir')
['lab1.py', 'lab1.exe', 'lab1.txt']
>>>
>>> glob.glob('/root/testdir/*.py')
['/root/testdir/lab1.py']
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *socket*

- The socket module is based on the BSD socket interface, but is available on all major platforms
- Supports IPv4 and IPv6
- Supports several protocols, including TCP and UDP
- Supports connecting to a remote host or acting as a server and accepting incoming connections
- For this class, we will focus on TCP over IPv4

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *socket* *(continued)*

- Example: connect to another socket on the loopback address:

```
>>> import socket
>>> sock = socket.socket(socket.AF_INET,
                         socket.SOCK_STREAM) # create a
socket
>>> remote_address = ('127.0.0.1', 50000) # addr/port of
remote service
>>> sock.connect(remote_address)
>>> sock.sendall(b'Hello world!\n')
>>> sock.recv(100) # listen for the server's response,
up to 100 bytes long
"The world says hi."
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *socket* *(continued)*

- Things to note:

LINE(S)	NOTE
>>> remoteAddress = ('127.0.0.1', 50000) >>> sock.connect(remoteAddress)	If there isn't a service listening at the foreign IP and port, we get an exception with the message, 'Connection refused'
>>> sock.sendall("Hello world!\n")	"send()" is supported as well, but isn't guaranteed to send all the bytes passed to it. Use sendall() to guarantee we send everything.
>>> sock.recv(100)	By default, the call to recv blocks until bytes are received

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *socket* *(continued)*

- Now to set up our own server!
- Create a socket, then bind to an interface and a port
- "listen(num)" tells the OS how many queued connections we allow on this socket; we start listening for new connections
- Accept() blocks, waiting for new connections. It returns a new socket for communicating with the client and the client's address and port

```
>>> sock = socket.socket()
>>> sock.bind(('127.0.0.1', 50000))
    # bind to an addr and a port
>>> sock.listen(1)
>>> sock_to_client, remote_address = sock.accept()
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *socket* *(continued)*

- How to close a socket: `>>> sock.close()`
- Changing the defaults:
  - Set a socket to non-blocking mode: `>>> sock.setblocking(0)`
  - If sock.recv(value) doesn't have bytes to return immediately (i.e. they were already buffered when the call was made), instead of blocking and waiting, an exception is thrown
  - We can also set a timeout: `>>> sock.settimeout(30)`
  - Now if sock.recv() or sock.send() can't complete immediately, they'll keep trying up to 30 seconds before throwing an exception

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python struct

- Converts Python values to C-structs (and back!)
- Use format strings to specify the types in the struct
  - 'B' = 1 byte, 'I' = 4 bytes signed, etc (there are many more!)
- Two directions: 'packing' - (Python values -> C struct)

  'unpacking' - (C -> Python)
  - Unpacking always results in a tuple
- Example:

```
>>> import struct
>>> a = 5
>>> struct.pack('B', a)
b'\x05'
```

- Read: "Pack an integer into a 1-byte string'

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python struct *(continued)*

- Endianness:
  - "<" means 'interpret as little endian'
  - ">" means 'interpret as big endian'
- Example:

```
>>> import struct
>>> a = b"\x01\x02\x03\x04"   # a struct we're going to unpack
>>> struct.unpack("<L", a)   # little-endian, unsigned long
(67305985,)
>>> struct.unpack(">L", a)   # big-endian, unsigned long
(16909060,)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python struct *(continued)*

- We can pack or unpack more than one value at a time:

```
>>> a = bytearray([0x01, 0x02, 0x03, 0x04])
>>> struct.unpack("BBBB", a) # unpack four 1-byte integers
(1, 2, 3, 4)
>>> struct.pack(">LBB", 5000, 4, 255) # a Long and two chars
b"\x00\x00\x13\x88\x04\xff"
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python random

- Gives you pseudo-random functions
- You'll probably be doing uniform selection from a given range. You can also do other distributions as well (normal, lognormal, etc).

```
>>> import random
>>> random.randint(0,10)
7
>>> nums = range(1,6)
>>> random.shuffle(nums)                    # shuffles in place
>>> nums
[4,1,2,5,3]
>>> for i in random.sample(nums, 3):
...    print(i)

5
2
3
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python StringIO

- Implements an in-memory file object that reads and writes a string buffer
- cStringIO is faster version of StringIO

```
>>> from io import StringIO
>>> msg = "This is a test"
>>> f = StringIO(msg)
>>> print(f.read())
This is a test

>>> f = StringIO()
>>> f.write("This is ")
>>> f.write("a test")
>>> print(f.getvalue())
This is a test
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python datetime, time

- Gives you access to date / time functions
- Gives you lots of options for manipulating and printing date / times

```
>>> import datetime
>>> print(datetime.datetime.now())
2016-03-05 13:22:39.319000

>>> import time
>>> time.sleep(5)

>>> d = datetime.datetime.now()
>>> d.strftime("%A, %B %d %Y at %H:%M:%S")
'Saturday, March 05 2016 at 13:22:44'
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *re* (Regular Expression Package)

- Regular expressions are a standardized way of searching, replacing and parsing text with complex patterns of chars
- Common Python *re* methods
  - *re.compile* – compiles a regular expression for repeated use
  - *re.match* – Try to apply the pattern at the start of the string, returning a match object, or None if no match was found
  - *re.search* – Scan through string looking for a match to the pattern, returning a match object, or None if no match was found
- Often simple string methods are enough to do what you need
- So use Python re module, but only when you need to!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *re* (Regular Expression Package)

```
>>> import re
>>> search_me = "searchme - abcdefg"
>>> my_re = re.compile("abcd")
>>> match_obj = my_re.search(search_me)
>>> if match_obj is not None:
...     print(search_me[match_obj.start():
                       match_obj.end()])
...
abcd
>>>
>>> match_obj2 = re.search("abcd", search_me)
>>> if match_obj2 is not None:
...     print(match_obj2.group(0))
...
abcd
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Regular Expression Mini-Lab

- I've given you a file named History.txt that contains the history of 7-Zip.
- Use Python and the re module to print out all the lines that contains a date
  - Determine the number of lines that contain a date
- Hint: If you're not a regular expression guru, helpme.txt contains one you can use.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *threading*

- The good:
  - Python threading is a limited threading implementation that allows users to abstract multiple threads of execution
  - Contains multiple synchronization constructs, including variants of locks and semaphores
- The bad:
  - Only one thread can ever run at a time in a Python process
    - Global Interpreter Lock (GIL) ensures this
      - per-python-process lock for interpreting bytecode

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *threading* *(continued)*

- Basic Usage
  - Thread creation:
    - thread_obj = threading.Thread(target=func, args=args, kwargs=kwargs)
  - Thread doesn't run until thread_obj.start()
  - To block the current thread until another thread finishes, use thread_obj.join()
  - Lock & Rlock
    - Both need paired acquire and release
    - RLock can be acquired multiple times in the same thread if it is released the same number of times ("Reentrant")
    - Lock would deadlock if it tried this, only acquire once

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *threading* *(continued)*

- Function-based Example

```
import threading

global_counter = 0
global_lock = threading.Lock()

def inc_counter(times=100, lock=False):
    global global_counter
    for i in range(times):
        if lock: # the 'right' way
            global_lock.acquire()
            global_counter += 1
            global_lock.release()
        else: # the 'wrong' way
            global_counter += 1
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *threading* *(continued)*

- Function-based Example

```
import threading


def run_threads(num_threads=10, *args, **kwargs):
    global global_counter
    global_counter = 0

    # Create a list that has num_threads number of threads in it
    threads = [threading.Thread(target=inc_counter, args=args,
                  kwargs=kwargs) for i in range(num_threads)]
    for t in threads: # start all threads
        t.start()
    for t in threads: # wait for all threads to finish
        t.join()
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *threading* *(continued)*

- Function-based Example

```
>>> run_threads(num_threads=10, times=1000, lock=True)
took 0.54  seconds
10000 should be 10000

>>> run_threads(num_threads=10, times=1000, lock=False)
took 0.05  seconds
4266 should be 10000
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *threading* *(continued)*

- Can also subclass threading.Thread
- Useful inspection stuff (good for debugging deadlocks):
  - Get thread objects using threading.enumerate()
  - Get current frames for a thread using sys._current_frames()[t.ident]
  - Can print current code location for frame with traceback.print_stack(frame) or save to list of strings with traceback.format_stack(frame)
  - So, putting it all together, to show:
    - for t in threading.enumerate():

      traceback.print_stack(sys._current_frames()[t.ident])

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Lab 8

Socket-y Goodness
All Packed Up

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- There is a server on the instructor's machine that holds a secret key.
  - In order to get this key, one must:
    - Connect to the server
    - Receive a pickled string that contains one byte of endianness and two unsigned longs
    - Un-pickle the string that contains the three values (endianness and longs)
    - Unpack the longs according to the one byte of endianness and sum them
    - Pack the sum and pickle it
    - Send the nicely wrapped package back to the server to find out the secret message
      - Hint: Receive some more data
- Don't hesitate to look up the socket module in the Python docs!!!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Tasks *(continued)*

- Write your own server to do what the instructor's server is doing.
- Use your client to connect to your server to verify it works.
- Bonus:
  - Make your server multi-threaded!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes*

- ctypes allows Python programs to call into C libraries (.dll on Windows, .so on Linux, .dylib on OS X, etc.)
- Allows developer to call C functions as if they were Python functions with minimal extra work
- Steps:
  - Load the library
  - Define what argument types the functions of interest take
  - Define the function's return type
    - Defaults to 'int'
  - Call the function

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Benefits to ctypes:
  - Interface with C libraries (possibly not designed for Python use)
  - Execute faster: code that must run very quickly can be written in C and called from Python
  - Faster development: once a prototype is developed, rather than develop the final version from scratch in C, only parts that need speed are rewritten and called from Python
  - Cross-platform: works on Windows, Linux, OS X, and others
  - True concurrency: ctypes releases the GIL before entering C functions, so other Python threads can execute while the C function executes

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Programmer sometimes needs to tell ctypes the types of the arguments and return value for each function
- Some example types.

ctypes TYPE	C TYPE	PYTHON TYPE
c_char	char	1-character string
c_wchar	wchar_t	1-character unicode string
c_short	(signed) short	int
c_void_p	void*	int or None
c_wchar_p	wchar_t* (NULL terminated)	unicode or None
c_double	double	float
c_ulong	unsigned long	int

- Note: there are many more, and you can define your own types too

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Example: (load a library and call a simple function)
- Let’s call remove. Pull up the man pages for remove!

```
>>> import ctypes
>>> libc = ctypes.CDLL('libc.so.6')
>>> remove = libc.remove
>>> remove.argtypes = [ctypes.c_char_p]
>>> remove.restype = ctypes.c_int # This is already int by default
>>> buff = ctypes.c_char_p(b"myfile.txt")
>>> ret_val = remove(buff)
>>> print(ret_val)
0
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- To call a function, we can define its argument types and, if return value isn't an int, its return type

```
>>> libc = ctypes.CDLL("libc.so.6")
>>> strchr = libc.strchr
>>> strchr.restype = ctypes.c_char_p
>>> strchr.argtypes = [ctypes.c_char_p, ctypes.c_char]
>>> strchr('abcdef', 'd')
'def'
>>> strchr('abcdef', 'def') # we send string, not char
…
ArgumentError: argument 2: exceptions.TypeError: one character
string expected
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Problem: sometimes dlls export functions with names that are not valid Python names
  - We can't do foodll.??2@YAPAXI@Z(arg)
- Answer: Use getattr

```
>>> myFunc = getattr(foodll, "??2@YAPAXI@Z")
>>> myFunc(arg)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Defining custom types: What if a function takes a pointer to a struct you define?
- Two steps: first define the struct:
  - struct classes must inherit from ctypes.Structure
  - _fields_ class property must be a list of two-tuples, each pair is a struct member name and any ctypes type

```
>>> # extend a ctypes Structure
>>> class Rect(ctypes.Structure):
        _fields_ = [("top_left_x",     ctypes.c_int),
                    ("top_left_y",     ctypes.c_int),
                    ("bottom_right_x", ctypes.c_int),
                    ("bottom_right_y", ctypes.c_int)]
>>> rect = Rect(10, 30, 40, 20)
>>> print(rect.top_left_x, rect.bottom_right_x)
10 40
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- We are trying to define our own struct, create an instance, and pass a pointer to it to a function
- Second step: make a pointer out of your object:
  - Note that Rect is a ctypes type because it extends ctypes.Structure
  - ctypes.pointer() only operates on ctypes types
- You may have to cast your pointers
  - Using the ctypes.cast() function

```
>>> rect = Rect(10, 30, 40, 20)
>>> pRect = ctypes.pointer(rect)
>>> dll.somefunc(pRect) # takes a pointer to a Rect struct
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Things to be cautious about, part 1:
  - Be careful when comparing ctypes values to each other
  - Why? In a ctypes type, accessing the contents results in the creation of a new Python object each time

```
>>> my_str= ctypes.c_char_p(b"abc def ghi")
>>> print(my_str.value)
'abc def ghi'
>>> my_str.value == my_str.value   # equality check
True
>>> my_str.value is my_str.value   # identity check
False
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Things to be cautious about, part 2:
  - Like with other ctypes objects, ctypes pointers create a new Python object with every access

```
>>> integer = ctypes.c_int(42)
>>> pInteger = ctypes.pointer(integer)
>>> pInteger.contents
c_long(42)
>>> pInteger.contents.value == pInteger.contents.value
True
>>> pInteger.contents == pInteger.contents
False
>>> pInteger.contents is pInteger.contents
False
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Things to be cautious about, part 3:
  - Python strings are immutable, so passing them into c functions that expect mutable strings is bad
  - To illustrate:

```
>>> s = "Hello world"              #  define initial string
>>> c_s = ctypes.c_char_p(s)
>>> print(c_s)
c_char_p('Hello world')
>>> c_s.value = 'New Example'   #  save a new string value to object
>>> print(c_s)
c_char_p('New Example')
>>> print(s)                       #  original string value is unchanged
Hello world
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Things to be cautious about, part 3 continued
  - To fix string-immutability issue, use ctypes.create_string_buffer()
  - Returns a buffer of mutable data that can be accessed either as raw data or as a null-terminated string:

```
>>> buff = ctypes.create_string_buffer(b"Hello")
>>> print(ctypes.sizeof(buff), repr(buff.raw))
6  'Hello\x00'
>>> print(buff.value) # access it as a null-terminated string
'Hello'
>>> buff.value = b'Hi'
>>> print(repr(buff.raw))
'Hi\x00lo\x00'    # Hi (null) overwrote the 'Hel' – then we have 'lo\x00'
leftover
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- Things to be cautious about, part 3 continued
  - ctypes.create_string_buffer can also take a number of bytes (example below)
  - create_unicode_buffer also exists to work with wide character strings

```
>>> buff = ctypes.create_string_buffer(4)
>>> print(repr(buff.raw))
'\x00\x00\x00\x00'
>>> buff.value = b'Hi'
>>> buff.raw
'Hi\x00\x00'
>>> buff.value
'Hi'
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ctypes* *(continued)*

- *ctypes* Summary:
  - We've only touched the surface – there is a lot of cool functionality there (unions, arrays, Python callback functions, etc)
  - You must be extra careful in ctypes – deref’ing a bad pointer can cause the entire Python process to crash (we don't just get a Python exception)
  - The ctypes doc on python.org has lots of good examples including a 'Surprises' section to demonstrate behavior that may not be intuitive – check it out

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Lab 9

ctypes awesomeness

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- Load the given dll/so
- Call the exported function MyMessageBox
  - MyMessageBox(const char *msg, const char *title);

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *(parsing command line args)*

- For quick command line parsing, use *sys.argv*:

```
import sys
print(sys.argv)
print('Arg0: "{0}"'.format(sys.argv[0]))
```

```
$ python script.py arg1
['script.py', 'arg1']
Arg0: "script.py"
```

- Additionally, you can use argparse!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *argparse*

- argparse provides a more object-oriented way of parsing args
- Available in Python 2.7+
- Program long and short options the same way
- Auto-generates 'help' printout when user types 'python <script> -h'
- Result values are stored to an object where they can be easily accessed – don't create a new variable for each option

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *argparse*, continued

- Example:

```
from argparse import ArgumentParser
parser = ArgumentParser()
parser.add_argument("-f", "–file", dest='filename',
help="Write to FILE", metavar="FILE")
parser.add_argument("-q", action='store_false',
dest='verbose', default=True, help='Suppress output')
args = parser.parse_args()
print(args.filename, args.verbose)
```

- Script:

```
$ python script.py –f myfile –q
myfile False
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *shutil*

- Allows copying/moving/deleting of populated directories
  - shutil.copytree()
    - Recursively copy an entire directory tree
    - Destination directory must not already exist
  - shutil.move()
    - Move a file or directory to another location
    - Destination should not already exist
  - shutil.rmtree()
    - Delete an entire directory tree

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *ConfigParser*

- Allows you to set up simple configuration files
- Sections defined to allow division of data
- Example

```
# config.conf
[Global]
username = clarkkent
```

```
import ConfigParser
config = ConfigParser.ConfigParser()
config.read("config.conf")
username = config.get('Global', 'username')
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python Logging

- Establishes several logging levels – debug, info, warning, etc
- Several built-in logging outputs
  - stdout, file, socket, and more!
- Programmatic or config-file based configuration
- Specify different formatting options
  - Process, thread, time, etc

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python Unittest

- Allows for programmatic validation of code
  - Validation based on assertions
- Sometimes referred to as "PyUnit"
- Very extendable
  - Can create custom assertions
- One of the biggest use cases of Python!

```
import unittest

class TestClass(unittest.TestCase):

    def test_adder_success(self):
        self.assertEqual(4 + 4, 8)

    @unittest.expectedFailure
    def test_adder_fail(self):
        self.assertEqual(4 + 3, 8)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python Base64

- Common way to encode/decode a string (also supports base16 and base32)
  - Using the module

```
import base64

>>> x = base64.b64encode(b'test')
>>> print(x)
b'dGVzdA==' ## padding
>>> print(base64.b64decode(x))
b'test'
```

  - Using the string method

```
>>> base64.encodebytes(b'test')
'dGVzdA==\n' ## trailing newline
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Other Random Libraries

- *Py2Exe (run Python scripts as executables)*
- *math (exponents, trig functions, constants, etc).*
- zlib, gzip, tarfile, zipfile (Compression)
- httplib, urllib, SimpleHTTPServer (web services)
- timeit, profile, pstats (Performance Measurement)
- dumbdb, bsddb, dbm, gdbm, sqlite3 (database access)
- xml, htmllib, json, mimify, binascii (data conversion)
- pylint, black (linter/formatters)
- LOTS AND LOTS MORE!

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Python has a package for data serialization called *pickle*
- Remember, some things can't be pickled
  - Locks, sockets, nested and lambda functions contain unpickleable objects
- Python's socket module is very powerful and based on the standard BSD socket interface
  - Supports IPV4 and IPV6
  - Does pretty much everything you would expect a socket library to do
- Python's *struct* module converts Python values to C-structs, but be wary of endianness

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review *(continued)*

- Python's *re* (regular expression) module allows standardized use of regular expressions
- Python's *threading* module allows users to abstract multiple threads of execution
  - Contains multiple synchronization constructs
  - BUT only one thread can run at a time based on the GIL
- Python's *ctypes* module allows Python to call into arbitrary C libraries

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review *(continued)*

- Two methods for parsing command line arguments:
  - sys.argv – useful for quick command line parsing
  - argparse – object-oriented method of parsing command line
- ConfigParser can be used to parse/modify config files
- shutil can come in handy when manipulating directories
- Questions?

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ADVANCED CYBER TRAINING PROGRAM

System Interfaces

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Use available system interfaces in Python.

- Enabling Objectives
  - Describe available system interfaces
  - Use sys, os, and subprocess to accomplish basic system tasks

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

Python *sys*

- The *sys* module contains system-level information, such as the version of Python you're running and system-level options such as the maximum allowed recursion depth
- Available methods in the *sys* module are platform dependent
- Useful *sys* methods/objects:
  - *sys.path* – list containing search path for modules and packages
  - *sys.argv*– list containing arguments to the Python executable
  - *sys.modules* – dictionary containing all loaded modules
- Explore by doing:

```
>>> import sys
>>> dir(sys)
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *os*

- Provides a portable way of using operating system dependent functionality
- The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface
- Useful *os* methods/objects:
  - *os.path* – package containing path manipulation methods
  - *os.walk* – walks a given directory returning a generator function
  - *os.system* – executes a given command
  - *os.stat* – performs a stat system call on a given path

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *subprocess*

- Allows you to spawn processes, connect to their input/ output/error pipes, and obtain their return codes
- Intended to replace functionality of:
  - *os.system*
  - *os.spawn*
  - *os.popen*

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *subprocess* *(continued)*

- Prior to Python 3.5, there were 3 main functions to use within *subprocess*
- subprocess.call(*args*, ***, *stdin=None*, *stdout=None*, *stderr=None*, *shell=False*, *cwd=None*, *timeout=None*)
  - Run the command with arguments. Wait for command to complete, then return the returncode attribute
- subprocess.check_call(*args*, ***, *stdin=None*, *stdout=None*, *stderr=None*, *shell=False*, *cwd=None*, *timeout=None*)
  - Run command with arguments. Wait for command to complete. If the return code was zero then return, otherwise raise CalledProcessError
- subprocess.check_output(*args*, ***, *stdin=None*, *stderr=None*, *shell=False*, *cwd=None*, *encoding=None*, *errors=None*, *universal_newlines=None*, *timeout=None*, *text=None*)
  - Run command with arguments and return its output. If the return code was non-zero, it raises a CalledProcessError.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *subprocess* *(continued)*

- In Python 3.5, the run() function was added to *subprocess*
- subprocess.run(*args*, ***, *stdin=None*, *input=None*, *stdout=None*, *stderr=None*, *capture_output=False*, *shell=False*, *cwd=None*, *timeout=None*, *check=False*, *encoding=None*, *errors=None*, *text=None*, *env=None*, *universal_newlines=None*)
  - Run command with arguments. Wait for command to complete, then return a CompletedProcess instance.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Python *subprocess* *(continued)*

```
>>> import subprocess
>>> subprocess.run(["ls", "-l"]) # doesn't capture output
CompletedProcess(args=['ls', '-l'], returncode=0)

>>> proc = subprocess.run("tasklist", capture_output=True)
>>> type(proc)
<class ‘subprocess.CompletedProcess’>
>>> proc.returncode
0

>>> subprocess.run("exit 1", shell=True, check=True)
Traceback (most recent call last):
  ...
subprocess.CalledProcessError: Command 'exit 1' returned non-zero exit status 1
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

Lab 10

subprocess madness

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Tasks

- Use the Python subprocess module to start and kill an application.
- Use the Python subprocess module to start an instance of tasklist/ps, pipe the output to a file, and display ONLY the PIDs and process names of all the running processes.
- Do the above command without the intermediate file
  - Hint: Look at the subprocess docs!
- Write a function that takes as input a PID, and use the subprocess module and taskkill/kill to kill the process with the given ID.
  - Bonus: Write another function that takes as input a process name and uses subprocess and taskkill/kill to kill ALL processes that have that name

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review

- Python's sys module contains useful system-level information and several useful objects/methods/sub-modules:
  - *sys.path*
  - *sys.argv*
  - *sys.modules*
- The os module provides portable access to operating system dependent functionality including:
  - *os.path*
  - *os.walk*
  - *os.stat*

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Lesson Review *(continued)*

ACTP
ADVANCED CYBER TRAINING PROGRAM

- Python *subprocess* allows you to spawn processes, connect to input/output/error pipes and obtain results and return codes
- Questions?

ManTech

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ADVANCED CYBER TRAINING PROGRAM

Python Gotchas

Mutable Default Arguments

- Python's default arguments are evaluated once when the function is defined
  - Use None instead!

```
def append_to(element, to=[]):
    to.append(element)

    return to
```

What You Might Expect

```
>>> print(append_to(12))
[12]
>>> print(append_to(42))
[42]
```

What You Get

```
>>> print(append_to(12))
[12]
>>> print(append_to(42))
[12, 42]
```

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Modifying While Iterating

- Never delete an element from a list while iterating over it
  - Temp list
  - List comprehension

```
>>> odd = lambda x : bool(x % 2)
>>> numbers = [n for n in
range(10)]
>>> for i in range(len(numbers)):
...   if odd(numbers[i]):
...     del numbers[i]
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
IndexError: list index out of range
```

```
>>> odd = lambda x : bool(x % 2)
>>> numbers = [n for n in
range(10)]
>>> numbers = [n for n in numbers
if not odd(n)]
>>> numbers
[0, 2, 4, 6, 8]
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Series Conclusion

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

Series Review

We've covered a lot of topics:

- Python Interpreter
- Types and Operators
- Statements
- Functions
- Modules/Packages
- Classes
- Exceptions
- Built-in Tools
- System Interfaces

ManTech
Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Series Assessment

- Python Test
- Python Series Evaluation

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

ADVANCED CYBER TRAINING PROGRAM

Python Mini Crucible

What Have We Learn

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Python Extras

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

multiprocessing

- Supports spawning processes with a similar API to threading module
- Can take advantage of all the processors on a given machine
- Arguments to worker function must be pickleable

```
import multiprocessing

def worker():
    """worker function"""
    print('Worker')
    return

if __name__ == '__main__':
    jobs = []
    for i in range(5):
        p = multiprocessing.Process(target=worker)
        jobs.append(p)
        p.start()
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Web Scraping

- Several ways to scrape websites in Python
- *urllib and urllib2* (built-in)
  - File-like way of manipulating pages given a URL
  - Can handle basic authentication, cookies, proxies, etc
- *requests* (not built-in)
  - Simple to use API
  - Don't have to encode parameters
  - Sessions with cookie persistence

ManTech

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Web Scraping *(continued)*

- Urllib2

```
>>> import urllib2
>>> response = urllib2.urlopen("https://www.google.com")
>>> response.code
200
>>> dir(response)
['__doc__', '__init__', '__iter__', '__module__', '__repr__', 'close', 'code', 'fileno', 'fp', 'getcode', 'geturl', 'headers', 'info', 'msg', 'next', 'read',
'readline', 'readlines', 'url']
```

- Requests

```
>>> import requests
>>> response = requests.get("https://www.google.com")
>>> response.status_code
200
>>> dir(response)
['__attrs__', '__bool__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__',
'__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__nonzero__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'_content', '_content_consumed', '_next', 'apparent_encoding', 'close', 'connection', 'content', 'cookies', 'elapsed', 'encoding', 'headers', 'history',
'is_permanent_redirect', 'is_redirect', 'iter_content', 'iter_lines', 'json', 'links', 'next', 'ok', 'raise_for_status', 'raw', 'reason', 'request', 'status_code',
'text', 'url']
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Web Scraping *(continued)*

- BeautifulSoup
  - Provides methods for parsing a tree
    - HTML, XML, etc
  - Allows you to easily find all URLs/images/etc on an HTML page

```
html_doc = """<html><head><title>The
Dormouse's story</title></head> <body> <p
class="title"><b>The Dormouse's
story</b></p> <p class="story">Once upon a
time there were three little sisters; and their
names were <a
href="http://example.com/elsie"
class="sister" id="link1">Elsie</a>, <a
href="http://example.com/lacie"
class="sister" id="link2">Lacie</a> and <a
href="http://example.com/tillie"
class="sister" id="link3">Tillie</a>; and they
lived at the bottom of a well.</p> <p
class="story">...</p> """
```

```
>>> from bs4 import BeautifulSoup
>>> soup = BeautifulSoup(html_doc)
>>> soup.title
<title>The Dormouse's story</title>
>>> for a_tag in soup.find_all('a'):
...     print(a_tag.get('href'))
...
http://example.com/elsie
http://example.com/lacie
http://example.com/tillie
```

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019

Web Scraping Mini-lab

- Using the web scraping tools I just showed you, parse the website I will provide and:
  - Download all the images (img) locally
  - Bonus: Write all the comments to a file

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905
Proprietary Information © Copyright 2019