SECURE, PARALLEL AND DISTRIBUTED COMPUTING WITH PYTHON

Lecutue 1: Introduction

INSTRUCTOR'S EXPECTATIONS

Reading

- Please read through the specifications a couple of times to understand the requirements before asking questions.
- Most of the assignments/ problem statements will be long. Jumping the gun without reading the whole thing could be detrimental.

INSTRUCTOR'S EXPECTATIONS

Basic Arithmetic/CS Knowledge

- The class includes implementing encryption algorithms, which involves understanding basic mathematical concepts – including binary numbers, bitwise operations, bit masking, modular arithmetic, etc.
- You are being forewarned. Math is not scary.
- If you need more information about these concepts, please ask the instructor or the TA as soon as possible.

INSTRUCTOR'S EXPECTATIONS

Initiative

- Try a few different approaches before asking for help.
- This is not an introductory class. You will be expected to accomplish certain things on your own.
- You will be given a week to 10 days for homeworks. Please start early. You need that amount of time to complete them.

INSTRUCTOR'S EXPECTATIONS

- Attendance
- The class is very incremental. So, skipping a few classes will get you into trouble. You are expected to attend class.
- While we understand that sometimes, circumstances result in missing a couple of classes, missing quite a few classes is not condoned.
- Binge learning is not recommended, and will not be helpful.

INSTRUCTOR'S EXPECTATIONS

- Effort
- You need to devote time outside class to practice. Practice is the only way to better yourself as a programmer.
- The instructor and the TA are available to help.
- Please do not hesitate to ask for
- help.

THIS IS NOT A PYTHON PROGRAMMING COURSE

- The purpose of the course is to give students a grounding in computer security, information management, parallel and distributed computing.
- Python was chosen as the programming language of choice, since it is syntactically easy, allowing you to concentrate on the core concepts of the course.
- You do not need Python experience, the class will give you a brief introduction into Python syntax, and introduce some important libraries.
- The tests will focus on the core concepts of the course.

ABOUT PYTHON

- Development started in the 1980’s by Guido van Rossum.
- Only became popular in the last decade or so.
- Python 2.x currently dominates, but Python 3.x is the future of Python.
- Interpreted, very-high-level programming language. Supports a multitude of programming paradigms.
- OOP, functional, procedural, logic, structured, etc.
- General purpose.
- Very comprehensive standard library includes numeric modules, crypto services, OS interfaces, networking modules, GUI support, development tools, etc.

NOTABLE FEATURES

- Easy to learn.
- Supports quick development.
- Cross-platform.
- Open Source.
- Extensible.
- Embeddable.
- Large standard library and active community.
- Useful for a wide variety of applications.

GETTING STARTED

- Before we can begin, we need to actually install Python!
- The first thing you should do is download and install a virtual machine.
- We will be using an Ubuntu virtual machine in this course. All instructions and examples will target this environment – this will make your life much easier.
- Do not put this off until your first assignment is due!

GETTING STARTED

- Choose and install an editor.
- For Linux, I use the command line + gvim
- If you would prefer to use an IDE, I recommend PyCharm (available for all platforms).
- Windows users will likely use Idle by default.
- Other editor options include vim, emacs, Notepad++, SublimeText, Eclipse, etc.
- Throughout this course, I will be using an Ubuntu environment for all of the demos involving 3rd party libraries.
- The TA's will be grading by running your program from the command line in an Ubuntu environment. Please test using something similar if you're using an IDE.

INTERPRETER

- The standard implementation of Python is interpreted.
- The interpreter translates Python code into bytecode, and this bytecode is executed by the Python VM (similar to Java).
- Two modes: normal and interactive.
- Normal mode: entire .py files are provided to the interpreter.
- Interactive mode: read-eval-print loop (REPL) executes statements piecewise.

INTERPRETER – NORMAL MODE

- Let's write our first Python program!
- In our favorite editor, let's create helloworld.py with the following contents:

```
print ("Hello, World!")
```

- On the terminal:

```
$ python3 helloworld.py
Hello, World!
```

- Note: In Python 2.x, print is a statement. In Python 3.x, it is a function. If you are using Python 2.x and want to get into the 3.x habit, include at the beginning:

```
from __future__ import print_function
```

- Now, you can write

```
print ("Hello, World!")
```

INTERPRETER – INTERACTIVE MODE

- Let's accomplish the same task (and more) in interactive mode.
- Some options:

  -c : executes single command.

  -O: use basic optimizations.

  -d: debugging info

```
$ python3
>>> print ("Hello, World!")
Hello, World!
>>> hellostring = "Hello,  World!"
>>> hellostring
'Hello, World!'
>>> 2*5
10
>>> 2*hellostring
'Hello, World!Hello, World!'
>>> for i in range(0,3):
       print ("Hello, World!")

Hello, World!
Hello, World!
Hello, World!

>>> exit()
```

SOME FUNDAMENTALS

- Whitespace is significant in Python. Where other languages may use {} or (), Python uses indentation to denote code blocks.
- Comments
    - Single-line comments denoted by #.
    - Multi-line comments begin and end with three "s.
    - Typically, multi-line comments are meant for documentation.
- Comments should express information that cannot be expressed in code – do not restate code.

```
# here's a comment
for i in range(0,3):

    print (i)

def myfunc():
    """here's a comment about    the
    myfunc function"""
    print ("In a function!")
```

PYTHON TYPING

- Python is a strongly, dynamically typed language.
- Strong Typing
- Obviously, Python isn't performing static type checking, but it does prevent mixing operations between mismatched types.
- Explicit conversions are required in order to mix types.
- Example: 2 + ''four" - not going to fly
- Dynamic Typing
- All type checking is done at runtime.
- No need to declare a variable or give it a type before use.

- Let's start by looking at Python's built-in data types.

NUMERIC TYPES

- The subtypes are int, long, float and complex.
- Their respective constructors are int(), long(), float(), and complex().
- All numeric types, except complex, support the typical numeric operations you'd expect to find.
- Mixed arithmetic is supported, with the "narrower" type widened to that of the other. The same rule is used for mixed comparisons.

NUMERIC TYPES

- int: equivalent to C's long in 2.x but unlimited in 3.x.
- float: equivalent to C's doubles.
- long: unlimited in 2.x and unavailable in 3.x.
- complex: complex numbers.
- Supported operations include constructors (i.e. int(3)), arithmetic, negation, modulus, absolute value, exponentiation, etc.

```
$ python
>>> 3 + 2
5
>>> 18 % 5
3
>>> abs(-7)
7
>>> float(9)
9.0
>>> int(5.3)
5
>>> complex(1,2)
(1+2j)
>>> 2 ** 8
256
```

SEQUENCE DATA TYPES

- There are seven sequence subtypes: strings, Unicode strings, lists, tuples, bytearrays, buffers, and xrange objects.
- All data types support arrays of objects but with varying limitations.
- The most commonly used sequence data types are strings, lists, and tuples. The xrange data type finds common use in the construction of enumeration- controlled loops. The others are used less commonly.

SEQUENCE TYPES - STRINGS

- Created by simply enclosing characters in either
  - single- or double-quotes. It’s enough to simply assign
  - the string to a variable.
- Strings are immutable.
- There are a tremendous amount of built-in string methods.

```
mystring = "Hi, I'm a string!"
```

SEQUENCE TYPES - STRINGS

- Python supports a number of escape sequences such as ‘\t’, ‘\n’, etc.
- Placing ‘r’ before a string will yield its raw value.
- There is a string formatting operator ‘%’ similar to C. A list of string formatting symbols is available in documentation.
- Two string literals beside one another are automatically concatenated together.

```
print ("\tHello,\n")

print (r"\tWorld!\n")

print ("Python is " + "so cool.")
```

SEQUENCE TYPES – UNICODE STRINGS

- Unicode strings can be used to store and manipulate Unicode data.
- As simple as creating a normal string (just put a 'u' on it!).
- Use Unicode-Escape encoding for special characters.
- Also has a raw mode, use 'ur' as a prefix.
- To translate to a regular string, use the.encode() method.
- To translate from a regular string to Unicode, use the unicode() function.

```
myunicodestr1 = u"Hi Class!"
myunicodestr2 = u"Hi\u0020Class!"
print (myunicodestr1, myunicodestr2)
newunicode = u'\xe4\xf6\xfc'
print (newunicode)
newstr = newunicode.encode('utf-8')
print (newstr)
print (unicode(newstr, 'utf-8'))
```

Output:

Hi Class! Hi Class!
äöü
äöü
äöü

SEQUENCE TYPES - LISTS

- Lists are an incredibly useful compound data type
- Lists can be initialized by the constructor, or with a bracket structure containing 0 or more elements.
- Lists are mutable – it is possible to change their contents. They contain the additional mutable operations.
- Lists are nestable. Feel free to create lists of lists of lists…

```
mylist = [42, 'apple', u'unicode apple', 5234656]
print (mylist)
mylist[2] = 'banana'
print (mylist)
mylist[3] = [['item1', 'item2'], ['item3', 'item4']]
print (mylist)
mylist.sort()
print (mylist)
print (mylist.pop())
mynewlist = [x*2 for x in range(0,5)]  print
(mynewlist)
```

Output:

[42, 'apple', u'unicode apple', 5234656]

[42, 'apple', 'banana', 5234656]

[42, 'apple', 'banana', [['item1', 'item2'], ['item3', 'item4']]]
[42, [['item1', 'item2'], ['item3', 'item4']], 'apple', 'banana'] banana
[0, 2, 4, 6, 8]

SEQUENCE DATA TYPES

- str: string, represented as a sequence of 8-bit characters
- unicode: stores an abstract sequence of code points.
- list: a compound, mutable data type that can hold items of varying types.
- tuple: a compound, immutable data type that can hold items of varying types. Comma separated items surrounded by parentheses.
- a few more – we'll coverthem later.

```
$ python
>>> mylist = ["spam", "eggs", "toast"] # List of strings!
>>> "eggs" in mylist
True
>>> len(mylist)
3
>>> mynewlist = ["coffee", "tea"]
>>> mylist + mynewlist
['spam', 'eggs', 'toast', 'coffee', 'tea']
>>> mytuple = tuple(mynewlist)
>>> mytuple
('coffee', 'tea')
>>> mytuple.index("tea")
1
>>> mylonglist = ['spam', 'eggs', 'toast', 'coffee', 'tea']
>>> mylonglist[2:4]
['toast', 'coffee']
```

COMMON SEQUENCE OPERATIONS

Operation	Result
`x in s`	Trueif an item of sisequal to x, else False.
`x not in s`	False if an item of sisequal to x, elseTrue.
`s + t`	Theconcatenation of sand t.
`s * n, n * s`	nshallow copies of sconcatenated.
`s[i]`	ith item of s,origin 0.
`s[i:j]`	Slice of sfrom i to j.
`s[i:j:k]`	Slice of sfrom i to j with step k.
`len(s)`	Length of s.
`min(s)`	Smallest item of s.
`max(s)`	Largest item of s.
`s.index(x)`	Index of the first occurrence of x in s.
`s.count(x)`	Total number of occurrences of x ins.

COMMON SEQUENCE OPERATIONS

- Mutable sequence types further support the following 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 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)`	Add x to the end of s.

COMMON SEQUENCE OPERATIONS

- Mutable sequence types further support the following operations.

`s.extend(x)`	Appends the contents of x to s.
`s.count(x)`	Return number of i's for which s[i] == x.
`s.index(x[, i[, j]])`	Return smallest k such that s[k] == x and i <= k <j.
`s.insert(i, x)`	Insert x at position i.
`s.pop([i])`	Same as x = s[i]; del s[i]; return x.
`s.remove(x)`	Same as del s[s.index(x)].
`s.reverse()`	Reverses the items of s in place.
`s.sort([cmp[, key[, reverse]]])`	Sort the items of s in place.

BASIC BUILT-IN DATA TYPES - SET

- set: an unordered collection of unique objects.
- frozenset: an immutable version of set.

```
>>> basket = ['apple', 'orange', 'apple', 'pear',  'orange']
>>> fruit = set(basket)
>>> fruit
set(['orange', 'pear', 'apple'])
>>> 'orange' in fruit  True
>>> 'crabgrass' in fruit  False
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b
set(['r', 'd', 'b'])
>>> a | b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
```

BASIC BUILT-IN DATA TYPES - DICTS

- dict: hash tables, maps a set of keys to arbitrary objects.

```
>>> gradebook = dict()
>>> gradebook['Susan Student']  = 87.0
>>> gradebook
{'Susan Student': 87.0}
>>> gradebook['Peter Pupil'] =  94.0
>>> gradebook.keys()
['Peter Pupil', 'Susan Student']
>>> gradebook.values()
[94.0, 87.0]
>>> gradebook.has_key('Tina Tenderfoot')
False
>>> gradebook['Tina Tenderfoot'] = 99.9
>>> gradebook
{'Peter Pupil': 94.0, 'Susan Student': 87.0,  'Tina Tenderfoot': 99.9}
>>> gradebook['Tina Tenderfoot'] = [99.9, 95.7]
>>> gradebook
{'Peter Pupil': 94.0, 'Susan Student': 87.0, 'Tina Tenderfoot': [99.9,   95.7]}
```

PYTHON INPUT AND CONTROL FLOW

- So now we've seen some interesting Python data types. Notably, we're very familiar with numeric types, strings, and lists.
- Input in Python is done with the input() function. It can take a string prompt as a parameter and returns a string. If we need to store th input as a different type, we would have to cast it.
- Eg:

```
X = int(intput("enter a number: "))
```

- That's not enough to create a useful program, so let's get some control flow tools under our belt.

CONTROL FLOW TOOLS

While loops have the following general structure.

```
while expression:
    statements
```

- Here, statements refers to one or more lines of Python code.
- The conditional expression may be any expression, where any non-zero value is true.
- The loop iterates while the expression is true.
- Note: All the statements indented by the same amount after a programming construct are considered to be part of a single block of code.

```
i = 1
while i < 4:
    print (i)
    i = i + 1
flag = True
while flag and i < 8:
    print (flag, i)
    i = i + 1
```

```
1
2
3
True4
True5
True6
True7
```

CONTROL FLOW TOOLS

The if statement has the following general form.

```
if expression:
    statements
```

- If the boolean expression evaluates to True, the statements are executed.
- Otherwise, they are skipped entirely.

```
a = 1
b = 0
if a:
    print ("a is true!")
if not b:
    print ("b is false!")
if a and b:
    print ("a and b are true!")
if a or b:
    print ("a or b is true!")
```

a is true!
b is false!
a or b is true!

CONTROL FLOW TOOLS

You can also pair an else with an if statement.

```
if expression:
     statements
else:
     statements
```

The elif keyword can be used to specify an else if statement.

Furthermore, if statements may be nested within each other.

```
a = 1
b = 0
c = 2
if a > b:
    if a > c:
        print ("a is greatest")
    else:
        print ("c is greatest")
elif b > c:
    print ("b is greatest")
else:
    print ("c is greatest")
```

```
c is greatest
```

CONTROL FLOW TOOLS

- The for loop has the following general form.

```
for var in sequence:
    statements
```

- If a sequence contains an expression list, it is evaluated first.
- Then, the first item in the sequence is assigned to the iterating variable var.
- Next, the statements are executed.
- Each item in the sequence is assigned to var, and the statements are executed until the entire sequence is exhausted.
- For loops may be nested with other control flow tools such as while loops and if statements.

CONTROL FLOW TOOLS

- Python has two handy functions for creating a range of integers, typically used in for loops.
- These functions are range() and xrange(). xrange() is only available on python 2
- They both create a sequence of integers, but range() creates a list while xrange() creates an xrange object.
- Essentially, range() creates the list statically while xrange() will generate items in the list as they are needed. (python 2)
- Python 3 ranges are automatically xranges for larger sizes.
- We will explore this concept further.

CONTROL FLOW TOOLS

- There are four statements provided for manipulating loop structures.
- These are break, continue, pass, and else.
- break: terminates the current loop.
- continue: immediately begin the next iteration of the loop.
- pass: do nothing. Use when a statement is required syntactically.
- else: represents a set of statements that should execute when a loop terminates.

LET'S WRITE A PYTHON PROGRAM

- Ok, so we got some basics out of the way. Now, we can try to create a real program. I pulled a problem off of Project Euler. Let's have some fun.
- Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
- By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.