Computer Systems and Networks

ECPE 170 – University of the Pacific

Crash Dive into Python

Lab Schedule

Activities

- Today
  - Python crash course
  - Lab 11 & 12
- Last 2 days of class
  - Lab 12 – Network Programming

Assignments Due

- Sun Dec 1st
  - Lab 11 due by 11:59pm
- Tues Dec 10th
  - Lab 12 due by 11:59pm

Person of the Day: Guido van Rossum

- Author of the Python programming language
  - Self-appointed “Benevolent Dictator For Life”
- Chose the name because he was “in a slightly irreverent mood (and a big fan of *Monty Python's Flying Circus*)”
- Has worked in numerous organizations , including NIST, Google and Dropbox

Python

What is Python

- It is an interpreted language for scripting and many other uses
- Its features include:
  - Objects
  - Dynamic types
  - A rich set of libraries
  - Extensibility through C (for speed critical code)
- It is most notorious for its indentation rules, using whitespace or tabs (and it is *very picky*)

Python datatypes

- Python supports many of the datatypes from C or C++:
  - Integers, floats, strings, booleans
- In addition, later versions support other useful types:
  - Complex numbers
  - Sequences (tuples, lists)
  - Dictionaries
  - Sets
  - Bytes and bytearrays

Python Tuples

- A *tuple* is an immutable collection of objects
- Tuples are denoted by parenthesis

```
>>> t = (1,2,3)
(1,2,3)
>>> type(t)
<type 'tuple'>
```

- The objects in a tuple do not need to be of the same type

Python Lists

- A *list* is an mutable collection of objects
- Lists are denoted by square brackets

```
>>> l = [1.5,'a',(3,True)]
[1.5, 'a', (3, True)]
>>> type(l)
<type 'list'>
```

- Lists can be edited, sorted, chopped up…

Python Sequences

- Tuples and lists are both types of *sequences*: individual items can be accessed in various ways
- To access a particular item in a sequence:

```
>>> print (t[0],l[1])
1 a
```

- Sequences can also be access from the end using negative indices

```
>>> print (t[-2],l[-1])
2 (3, True)
```

Python Sequences

- Slices (subsets of sequences) are accessed by using a ":"

```
>>> t[0:2]
(1,2)
>>> l[1:]
['a', (3, True)]
```

- Note that the second index (if supplied) is one greater than actual last object in the slice

Python Dictionaries

- A *dictionary* is an associative array of keys and value pairs:

```
>>> d={'a':1, 'b':2, 3:'c'}
>>> d
{'b': 2, 3: 'c', 'a': 1}
>>> d.keys()
dict_keys(['b', 3, 'a'])
>>> d.values()
dict_values([2, 'c', 1])
>>> d['a']
1
>>> d['c']
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
KeyError: 'c‘
```

Python Error Handling

- Python handles errors using the `try` and `except` statements:

```
>>> try:
...     d['c']
... except:\
..     print ('bad key value')
...
bad key value
```

Python Blocks

- Python uses whitespace and ":" to denote blocks
  - WARNING: tabs and spaces are not interchangeable!
- Within a block, all lines are indented exactly the same amount
  -
```
>>> print (l)
[1.5, 'a', (3, True)]
>>>     print (l)
  File "<stdin>", line 1
        print (l)
        ^
IndentationError: unexpected indent
```

Python Statements and Flow Control

- Python supports these statements: `if, elif, else, for, while`

```
>>> if 1 > 2:
...   print (a)
... elif 3 > 2:
...   print (t)
... else:
...   print ('neither')
...
(1, 2, 3)
```

Python Statements and Flow Control

- The `for` statement takes a sequence as its input:
  -
```
>>> for x in (1,3,5,'a'):
...   print (x)
...
1
3
5
a
```

- This will also work for any sequence type (tuples, lists, strings, etc)

Python Statements and Flow Control

- For the equivalent of a C `for` loop, use the `range` class

```
>>> for i in range(0,9,3):
...  print (i)
...
0
3
6
```

- This is equivalent to

```
for (int i=0; i < 9; i += 3)
```

Using Python Libraries

- Libraries (*modules*) are accessed using the `import` statement

```
import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', 'acos',
    'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
    'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
    'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
    'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot',
    'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
    'log10', 'log1p', 'log2', 'modf', 'pi', 'pow',
    'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
    'trunc']
>>> math.sin(2)
0.9092974268256817
>>> from math import sin
>>> sin(2)
0.9092974268256817
```

The dir Function

- Within the interpreter, `dir` is handy for exploring variables and libraries

```
>>> t=[1.5,'a',(3,True)]
>>> type(t)
<class 'list'>
>>> dir(t)
['__add__', '__class__', '__contains__',
'__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__',
'__hash__', '__iadd__', '__imul__', '__init__',
'__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__',
'__sizeof__', '__str__', '__subclasshook__',
'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

- Anything with "__xxx__" is a built-in operation

Runtime evaluation

- Since Python is interpreted, and has dynamic typing, syntax is checked when code is first encountered, but variable types (or even their existence) aren't checked until the code is executed
- As a result, sometimes code will execute correctly for a while until either an undefined variable is encountered, or it is used incorrectly (i.e., trying to access an integer as a sequence)

The `struct` Module

The `struct` Module

- Since the details of variables are hidden in Python (for example, how many bytes is an integer?), there are no built-in ways to store values into files along with their encoding
  - A typical Python file would contain just ASCII or Unicode values
- The `struct` module deals with binary data
- In reality, it performs conversions between basic Python datatypes and binary strings

The `struct` Module

- There are two main functions in the `struct` module
  - `pack`: convert a group of variables into a string
  - `unpack`: convert a string into a group of variables
- These are similar to C's `printf` and `scanf`
- Each function requires a format string to describe how to pack or unpack the arguments

The `struct` Module

- Since we may need to convert data which is larger than one byte, endianness is an issue
- The first character of the format string determines the endianness

Character	Byte order	Size	Alignment
@	Native	Native	Native
=	Native	Standard	None
<	Little	Standard	None
>	Big	Standard	None
!	Network (Big)	standard	None