



Computer Systems and Networks

ECPE 170 – University of the Pacific

Crash Dive into Python

Lab Schedule

Activities

- **Today**
 - Python
 - Endianness
- **Tuesday**
 - Network programming
 - **Lab 8 – Network Programming**

Assignments Due

- **Lab 8**
 - **Due by Oct 26th 5:00am**
- **Lab 9**
 - **Due by Nov 2nd 5:00am**

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?

- Interpreted language for scripting and many other uses
- Features:
 - 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 datatypes from C or C++:
 - Integers, floats, strings, booleans

- Recent Python versions support other useful types:
 - Complex numbers
 - Sequences (tuples, lists)
 - Dictionaries
 - Sets
 - Bytes and bytearray

Runtime evaluation

- Python is interpreted and has dynamic typing
- Implications:
 - Syntax is checked when code is first encountered
 - Variable types (or even their existence) aren't checked until the code is executed
- Result: Code can 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)

Python Tuples

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

```
t = (1, 2, 3)
```

- 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)]
```

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:

```
t = (1, 2, 3)
l = [1.5, 'a', (3, True)]
print(t[0], l[1])
```

Output:

```
1 a
```

Python Sequences

- Sequences can also be accessed from the end (instead of beginning) using *negative* indices

```
t = (1, 2, 3)
l = [1.5, 'a', (3, True)]
print(t[-2], l[-1])
```

Output:

```
2 (3, True)
```

Python Sequences

- Slices (subsets of sequences) are accessed by using a “:”
- Note that the second index (if supplied) is one greater than actual last object in the slice

```
t = (1, 2, 3)
l = [1.5, 'a', (3, True)]
print(t[0:2])
print(l[1:])
```

Output:

```
(1, 2)
['a', (3, True)]
```

Python Dictionaries

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

```
d={'a':1, 'b':2, 3:'c'}  
print(d)  
print(d.keys())  
print(d.values())  
print(d['a'])  
print(d['c'])
```

Output:

```
{'a': 1, 3: 'c', 'b': 2}  
dict_keys(['a', 3, 'b'])  
dict_values([1, 'c', 2])  
1  
KeyError: 'c'
```

Python Error Handling

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

```
try:  
    d['c']  
except:  
    print("Key 'c' is not present")
```

Output:

```
Key 'c' is not present
```

Python Blocks

- Python uses whitespace and “:” to denote blocks
 - **Note: tabs and spaces are not interchangeable!**
- Within a block, all lines are indented exactly the same amount

```
print(1)
    print(1)
```

Output:

```
[1.5, 'a', (3, True)]
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")
```

Output:

```
(1, 2, 3)
```


Python Statements and Flow Control

- The `for` statement takes a sequence as its input
- This works for any sequence type
 - Tuples, lists, strings, etc...

```
for x in (1,3,5,'a'):  
    print(x)
```

Output:

```
1  
3  
5  
a
```

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)
```

Output:

```
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  
print(math.sin(2))
```

Output:

```
0.9092974268256817
```