Introduction to Python

EECS 4415
Big Data Systems

Tilemachos Pechlivanoglou
tipech@eecs.yorku.ca
Background
Why Python?

- "Scripting language"
- Very easy to learn
- Interactive front-end for C/C++ code
- Object-oriented
- Lots of libraries
  - including tools for data analysis
- Powerful, scalable
  - supporting tools that handle very large datasets
Pseudocode

```plaintext
if grade equals 60 and assignment in assignments list
   print "passed"
else
   print "failed"
```
```python
if grade == 60 and assignment in assignments_list:
    print("passed")
else:
    print("failed")
```
Python syntax

- Much of it is similar to C syntax
- Exceptions:
  - *missing operators:* `++`, `--`
  - *no {} for blocks*
    - only whitespace and indentation
  - *different keywords*
  - *no type declarations!*
  - *lots of extra features*
Starting and exiting Python

% python
Python 3.5.2 ...
>>> print("hello")
hello
>>> ^D
%

%
Running a Python file

Contents of file.py:

```python
print ("hello world")
```

Executing it in terminal:

```
% python file.py
hello world
%
```
Simple data types

- Numbers
  - integer
  - floating-point
  - complex!
- Strings
  - characters are strings of length 1
- Booleans are 0/1 (or False/True)
- Comments with #
Simple data types: operators

- + - * / % (like C)
- += -= etc. (no ++ or --)
- Assignment using =
  - but semantics are different!
    a = 1
    a = "foo" # OK
- Can also use + to concatenate strings
Python Script

Contents of file.py:

```python
print ("hello world")
```

Executing it in terminal:

```
% python file.py
hello world
%
```
Compound data types (1)

- Lists:

```
a = [1, 2, 3, 4, 5]
print (a[1])  # 2
some_list = []
some_list.append("foo")
some_list.append(12)
print (len(some_list))  # 2
```
Compound data types (2)

- **Tuples:**

```python
a = (1, 2, 3, 4, 5)
print(a[1])  # 2
empty_tuple = ()
```

- **Difference between lists and tuples:**
  - lists are mutable; tuples are immutable
  - lists can expand, tuples can’t
  - tuples are slightly faster
Dictionaries:

```
import this
a = {“age”: 18, “b”: “123a”, 3: True}
print (a[3]) # True
print (a[“age”]) # 18
```

Key-Value pairs
  - key can be number or string
  - value can be anything, including another sub-dictionary
Compound data types (4)

- **Objects:**

```python
class Thingy:
    # methods and properties
    t = Thingy()
    t.method()
    print (t.field)
```

- Built-in data structures (lists, dictionaries) also objects - *though internal representation is different*
Control flow (1)

- **if**, **if/else**, **if/elif/else**

```python
if a == 0:
    print ("zero!"")
elif a < 0:
    print ("negative!"")
else:
    print ("positive!"")
```

- **Notes:**
  - *blocks delimited by indentation!*
  - *colon (:) used at end of control flow keywords*
Control flow (2)

- **while** loops

```python
a = 10
while a > 0:
    print (a)
    a -= 1
```
Control flow (3)

- *for* loops

```python
for a in range(10):
    print (a)
```

- Really a "foreach" loop
- Common *for* idiom:

```python
a = [3, 1, 4, 1, 5, 9]
for i in range(len(a)):
    print (a[i])
```
Control flow (4)

- **pass** keyword

```python
if a == 0:
    pass  # do nothing
else:
    # whatever
```

- **continue** statement similar to C
File access

- **for..in** loops

```python
f = open("some_file", "r")
for line in f:
    # do something with line...
```

- Files normally end in .py
Functions

- **Definition**
  ```python
def foo(x):
    y = 10 * x + 2
    return y
  ```

- **Execution**
  ```python
  print(foo(10)) # 102
  ```

- All variables are local unless specified as `global`
- Arguments passed by value
Access other code by importing modules:

```python
import math
print (math.sqrt(2.0))
```

or

```python
from math import sqrt
print (sqrt(2.0))
```

or

```python
from math import *
import sys, string, math
```
Modules (2)

- Try to avoid 
  - dumps all names from `some_module` into local namespace
  - easy to get name conflicts this way

- Code you write in file `foo.py` is part of module "foo"
  
  ```python
  from foo import my_function
  import bar
  ```
Strings and formatting

```python
i = 10
d = 3.1415926
s = "I am a string!"
print ( "%d\t%f\t%s" % (i, d, s) )
```
Links and other material

- Python tutorials:
  - https://www.w3schools.com/python/
  - https://www.learnpython.org/

- Python documentation:
  - https://docs.python.org/3/