Programming in Python

CU Research Computing and CRDDS

Nick Featherstone
feathern@colorado.edu

Who are we?
Why are we here?

Should You Be Here?

- Target Audience:
  - (minimally) experienced programmers

Are you prepared?

- Did you already install the Intel Distribution for Python?
  - if not, we can chat after class

Course Outline

- Sep 21: python overview, variables
- Sep 28: conditionals, functions
- Oct 5: loops, lists
- Oct 12: objects
- Oct 19: NumPy
- Oct 26: HDF5, NetCDF, Pandas
- Nov 2: plotting with Matplotlib
- Nov 9: modules, integrating C and Fortran

Python is an Interpreted Language

- If you remember nothing else, remember this …
- Interpreters execute code “naively”
- They read your code line by line, and execute that code (more or less) exactly as you write it.
- This can lead to efficiency issues when compared against compiled code. Why?
- Compilers look to the future. Interpreters do not.

Compilers vs. Interpreters

Original Code

```
x =     2*a
x = x + 2*b
x = x + 2*c
```

Interpeted Code

```
x =     2*a
x = x + 2*b
x = x + 2*c
```

3 multiplies; 2 adds

Compiled Code

```
x = 2*( a + b + c)
```

1 multiply; 2 adds

For heavy-duty number-crunching, use NumPy (week 5)!

Our First Program

- Open your favorite text editor
- Type the following line in your file:

```
print("hello")
```

- Save the file as hello.py
- This is a complete python program
  - … no semicolons, no brackets
  - … no "begin program," no "end program," etc.
- Now let's run the program

Running Python Programs

There are various ways to modify and run python code…

- "*python program_name*"
- Interactive session
- ./program_name (unix/linux script style)
- Interactive Development Environment (IDE)
- Jupyter Notebook

…follow along as we try a few…

Invoking the Interpreter

- Run the program
  - Access your shell (“anaconda prompt” in windows)
  - Type: source activate idp (no source in windows)
  - Navigate to the folder containing hello.py
  - Type: “python hello.py”
- Common way of running Python programs

Running Code Interactively

- Similar to IDL or R environments
- Type one statement at a time & hit enter
- Type exit() when finished (exit is a function)
- Let's try it out

- Have existing code you want to run? Within interactive session:
  - exec(open("hello.py").read( ))
  - Note really standard practice

Running Code: Script-Style

- Can execute code in fashion similar to a bash script
- Provide “shebang” sign + path to python interpreter:

```
#! /usr/bin/python
print(“hello”)
```

- Try it (create hello2.py)
- Type: chmod +x hello2.py
- Type: ./hello2.py
- Note: the # symbol is Python’s comment symbol
- Note: /usr/bin/python is (probably) not your Intel python

Running Code: Script-Style

- We can check the python version within the script

```
#! /usr/bin/python
import sys
print(sys.version)
```

- Save this as ./hello3.py
- chmod + x hello3.py
- ./hello3.py
- *sys* is a *module* (collection of functions & variables)
- *version* is a variable within the sys module

Integrated Development Environment (IDE)

- Python always comes packaged with a simple IDE.
- Useful for highlighting function names etc.
- Run the program
  - Access your shell (“anaconda prompt” in windows)
  - Type: source activate idp (no source in windows)
  - Type: idle3 ← note the “L”
  - Follow along

Running Code: Jupyter Notebook

- Browser-integrated IDE
- VERY popular for interactive data-analysis
- Run the program
  - Access your shell (“anaconda prompt” in windows)
  - Type: source activate idp (no source in windows)
  - Type: jupyter notebook ← note the “Y”
  - Follow along

Program Structure in Python

- Customary to include “program” within a function
- Very helpful for complex and/or production codes

```
def main( ):
    print(“hello world”)

if __name__ == “__main__”:
    main( )
```

- Program is a function definition + function call
- Not necessary for our short exercises

Print Function Syntax

print(item1, item2, item3, …, sep = ‘ ‘, end= ‘\n’)

- item1, item2, item3
  - Comma-separated list of variables whose values you wish to display
- sep:
  - optional keyword parameter
  - separation string inserted between displayed values (defaults to whitespace)
- end:
  - optional keyword parameter
  - string appended to end of printed values (defaults to newline)

Using Print

- Try these different print combinations:

```
name = 'John'
age = 30
name2 = 'Mary'
age2 = 31
```

```
print(name, 'is', age, 'years old.')
print(name2, 'is', age2, 'years old.')
```

```
print(name, 'is', age, 'years old.' , end = ' ; ' )
print(name2, 'is', age2, 'years old.')
```

```
print(name, age, sep= ' : ')
print(name2, age2, sep = ' : ')
```

Variables in Python

- Variables are not declared
- Variables are created via an assignment statement
- Variable type determined implicitly via assignment
  - x = 2 (type “int”)
  - y = 3.0 (type “float”)
  - Z = “hello” (type “str” ; double or single quotes)
  - z = True (type Bool; note capital “T” ; “F” in False)
- Beware: Python is CASE SENSITIVE: z is not Z…
- Can check variable type using type function:
  - print(‘z is: ‘, type(z))

Arithmetic in Python

- Arithmetic in Python respects order of operations
- Addition: +
- Subtraction: -
- Multiplication: *
- Division: / (ALWAYS returns float result)
- Floor Division : // (Returns integer or float; rounds down)
- Mod (i.e., remainder) Division : % e.g, 3%2 → 1
- Exponentiation: ** e.g., 2**4 → 16
- Can concatenate strings using "+"
  - x = 'hello' + ' there'
  - print (x) → displays 'hello there'

Type Conversion

- Variables can be converted using type conversion functions
- x = int (43.4) → x is 43
- y = float (x) → y is 43.0
- z = str (x) → z is “43”
- n = bool (0) → n is False
- m = bool (x) → m is True

Variables and Memory

- Memory in python is a bit non-intuitive
- Characters and integers exist in one place in memory
- Can explore this using the "is" operator
  - True if variables point to *same memory location*
  - False otherwise
  - DOES NOT compare VALUES
- Try these:

```
a = 1
b = 1
print (a is b)
```

```
a = 1.0
b = 1.0
print (a is b)
```

```
a = 'T'
b = 'T'
print (a is b)
```

Variables and Memory

- Memory in python is a bit non-intuitive
- Intrinsic variables, like ‘int’ don’t occupy a set amount of RAM
- e.g., all ‘ints’ are not 4 bytes…
- Can explore this using the getsizeof operator
  - part of the sys module
  - returns size of an object in bytes
- Try these:

```
import sys
print( sys.getsizeof ( 2**30))
```

```
import sys
print( sys.getsizeof ( 2**60))
```

- To create standard X-byte datatypes, we need to use the NumPy package (efficiency gains; week 5)

Lists in Python

- Multiple values can be grouped into a list
  - mylist = [1, 2, 10]
- List elements accessed with [ ] notation
- Element numbering starts at 0
- print (mylist [1]) → displays 2
- Lists can contain different variable types
  - mylist = [1, ‘two’ , 10.0]
- Strings can be accessed element-wise like a list
  - mystring = ‘John’
  - print (mystring[1]) → displays ‘o’
- More on lists in two weeks…

Writing to Files in Python

```
#####################################
# generate some data
name1 = 'John' ; age1 = 30
name2 = 'Mary' ; age2 = 31
line1 = name1+' : ' + str(age1)
line2 = name2+' : ' + str(age2)

#####################################
#  write data to a file
filename = 'myfile.txt'
filemode = 'w'  # use 'w' when writing; 'r' when reading
file = open ( filename, filemode)
file.write(line1)    ;    file.write(line2)
file.close( )
```

multiple statements per line with semicolon