Programming Fundamentals with Python

Getting Started with Programming
Using the Interpreter

- Python is an interpreted language. It can be run directly from the command line by invoking `python`
  - On windows, it may be necessary to run `python.exe`
- To leave the interpreter, type `quit()` or `exit()`
- Programs can also be run from scripts by
  - `python myscript.py`
- Under Unix, the python command can be omitted by starting the script with
  - `#!/usr/bin/python`
Exercise

• Find the Python interpreter. It may have the IDLE development environment, which you may also use. Start the interpreter or the IDLE. The prompt is a triple > (>>>>).
Variables, Expressions, and Statements

- A variable can be an integer, a long (meaning unlimited) integer, or a double. There are no single-precision floats. Variables are not declared by type; the interpreter determines whether they are integer or float by the format. Names are case sensitive.
  - 1 36789 : Integers
  - 100000000000000L : Long integer
  - 1.314 6.022e23 : Floating point (double)
  - Python also supports complex doubles
    - 1.0+2.7j or 1.0+2.7J
- Boolean objects are available and take the values True and False (note capitalization)
Expressions

- Python supports the usual arithmetic operations
  
  + - * /

- Exponentiation (like Fortran's)
  
  **

- Truncating division
  
  //

  - Note: in Python 2.5 and below, 1/3=0. In Python 3.0, 1/3 will be converted to a double and will return the fractional part. But 1.0//3.0 is 0.

- Remainder
  
  %

- Comment
  
  Anything after # is ignored
Operator Precedence

- The evaluation of some statements is ambiguous unless an ordering is imposed. For example

  \[ z = 10. + 14. - 12. / 4. \]

- Operator precedence determines the evaluation. Expressions are evaluated from left to right in order *, /, +, -, so the above statement evaluates to 21 (i.e. 24-3).

- Parentheses can be used to change the default.

  \[ z = 10. + (14. - 12.) / 4. \]
Exercise

• At the prompt, type
  \[ \frac{1}{3} \]
  \[ \frac{1}{3} \]
  \[ \frac{1}{3} \]
  \[ \frac{1}{3} \]

  Compare the answers.

• Type some expressions. Use as many of the operators as you can.
Strings

- Strings are collections of characters. They are immutable, i.e. once assigned, operations cannot change them. Python does not have a single character type; a character is a one-element string.

- Strings are indicated by single quotes, double quotes, or triple quotes. Triple quotes """" indicate a block that is to be copied verbatim, including spaces and line breaks.

- Counting of characters in a string starts at 0.
Some String Operations

- Assignment
  
  ```python
  s='my string'
  s1="""Spam, spam, spam, sausage, eggs
      and spam."""
  ```

- Concatenation

  ```python
  s=s0+.txt
  ```

- Substring (slicing)

  ```python
  spart=s[3:5]
  ```

  - Note: the second value in the brackets is the upper bound non-inclusive, so this takes the third and fourth characters of s.

- Length

  ```python
  L = len(s)
  ```
Type Conversions (Casting)

• When mixed-type operations occur, the operands must be converted to be of the same type. This is called casting. Often Python will do this automatically; variables are promoted to the higher-ranked type. The rank is, from lowest to highest, integer, double, complex.

```python
>>> i=3*2
>>> i=6
>>> a=3.*2
>>> a=6.0
```
Explicit Casting

• Python provides functions for casting when needed.

  >>> a=float(3)/float(2)
  >>> i=int(a)

• Conversion to/from strings
  • Number to string, use the str() function
    >>> age=str(37)
  • String to number, use int or float
    >>> time=float('52.3')
Statements

● A statement is a complete “sentence” of the program; it describes some discrete action.

● Python statements do not require a semicolon at the end and it is considered bad form to add one.

● Multiple statements may be placed on a line if they are separated by semicolons.

● Backslash \ is the line-continuation character

● Lists and other data structures that are comma-separated may be continued on subsequent lines with the comma at the end of the lines.
Statement examples

• $x = x + 1$
  • Equivalent to $x += 1$
• $(x, y, z) = myfunc(a)$
• $f = open("myfile.txt", "w")$
• $x = 0; y = 1; z = 2$
• $A = [1, 2, 3, 4, 5, 6]$
More Advanced Data Structures

Lists

Python lists are ordered collections of objects. They may contain any types.

• Examples
  – L1=[“John”, “Eric”, 157, 92]

• Lists can contain other lists
  – L2=[’red’, ’blue’, [’green’, ’orange’]]

• Lists are mutable
  – L1[1]=”Terry”
    • L1=[“John”, ”Terry”, 157, 92]
List Operations

- **Slice**
  - L2=L1[0]; L3=L1[1:4]

- **Concatenate**
  - L4=[1, 2, 3]+[4, 5, 6]

- **Append**
  - L1.append("Graham")

- **Extend**
  - L1.extend(["Graham","Michael"])

- **Shorten**
  - Del L2[3]

- **Length**
  - LofL1=len(L1)
Some Useful Built-in Functions

- **reduce**(`func`, `S`)
  Collectively applies a function of two variables to sequence `S` and produces a single result. E.g.
  - `L=reduce(sum, a)`
    sums all the elements of `a`, when `sum` is defined as `x+y`.

- **map**(`func`, `S`)
  Applies the function to each element of `S` and returns a new list.
  - `L=map(square,S)`
  - Or
  - `L=map(lambda:x=x**2, S)`

- **filter**(`func`, `S`)
  Applies `func` to each element of `S`, returning True or False, and returns a new sequence consisting of all elements of `S` that are True.
  - `L=filter(lambda x: x>0, S)`
List Comprehension

- A list comprehension is a concise way to create a new list without using reduce, map, or filter. It is powerful but can be confusing.

- Syntax
  
  \[ \text{expression for } \text{var} \text{ in } \text{list if } \text{condition} \]
  
  The \textit{if} is optional.

- Examples
  
  \[ \text{x**2 for } \text{x in vector} \]
  
  \[ \text{sqrt(x) for } \text{x in vector if } \text{x > 0} \]
Exercise

- Make a list and a list comprehension
  
  `vector=[1, 2, 3, 4]`
  
  `vsquared=[x**2 for x in vector]`
  
  `vsquared`

- Notice that the comprehension expression is enclosed in square brackets. This is usually necessary.
Tuples

Like lists, tuples are ordered collections of objects.

- Indicated by parentheses
- Unlike lists, they are immutable (cannot be changed in place). So few list operations apply to them.
- Tuples can be nested arbitrarily
- Example
  - $T1=(a, b, c, d)$
Dictionary

- Dictionaries are unordered collections of objects. Items are stored and retrieved by key, not by position. They are written as key-value pairs within curly braces.

- Dictionaries are mutable and can be of any length (up to the limits of the system).

- Dictionaries can be nested.

- Example
  - D1={ 'bear':'panda','cat':'leopard','dog','wolf'}
Dictionary Operations

- Index (by key)
  \[ \text{Value} = D1[\text{'bear'}] \]
- Membership test
  \[ D1.\text{has\_key('dog')} \]
- List of keys
  \[ L1 = D1.\text{keys()} \]
- Length (number of entries)
  \[ \text{LofD} = \text{len}(D1) \]
Copying Simple Objects

• Assignment of an object to another variable merely produces a pointer to the original object.

```python
a=[0, 1, 2, 3, 4, 5]
b=a
b[2]=10
a=[0,1,10,3,4,5]
```

• For actual copies, use the copy function

```python
import copy
c=copy.copy(a)
```
Program Control Structures

We will now cover the basic program-control statements in Python. These include

- Conditionals
- Loops
- Functions

Flow-control structures are how your program actually performs interesting computations! They are based on blocks of code. In Python, blocks are indicated by indentation. You must indent all statements that make up a block by the same amount. You can choose the number of spaces (don't use tabs), but it must be the same within a block.
Conditionals

- Conditionals are blocks of code that are executed subject to some condition. The condition must evaluate to *true* or *false*.

- In Python, the Boolean values are True and False and the programmer does not need to associate them with numbers.

- Syntax

  ```python
  if <condition1>:
    block1
  
  elif <condition2>:  # optional
    block2
  
  else:               # optional
    block3
  ```
Conditional Operators

- Booleans can be evaluated with
  - `==` equality
  - `!=` inequality
  - `<`, `<=` less than, less than or equal to
  - `>`, `>=` greater than, greater than or equal to
  - `and`
  - `or`
  - `not`
    - Note that `and`, `or`, and `not` are spelled out
More Conditionals

- Single-line statements:
  ```
  if x=0: z=0
  ```
- There is no case statement; use `elif`
  ```
  If cond1:
    block1
  elif cond2:
    block2
  elif cond3:
    block3
  else:
    final choice
  ```
Loops

• Loops are used when repetitive blocks of code are to be executed in sequence.

• Examples
  • Read all the lines in a file
  • Perform a computation on every element of an array
While Loop

• The most general loop structure in Python is the while loop.

• Syntax

  while <condition>:
    block1
  else:  # optional
    block2

• The else is executed only if the loop ran to completion
While Example

while x<0 and y<0:
    z=0
else:
    z=1
Exercise

• Type at the prompt
  
  x=1

  while ( x < 10 ):
    x=x+1

  x

  <enter>

  If you are using IDLE, it will autoindent for you, so the enter is to cancel that.
For Loop

- The other fundamental loop structure is the for loop. In Python it takes the form

```python
for <iterated item> in <iterator object>:
    block1
else:  # optional
    block2
```
For Examples

for name in ["John", "Fred", "Jim", "Steve"]:
    mylist.append(name)
else:
    mylist.append("Tom")

- The range function is often used with for loops. It produces a list of integers from \textit{start} to \textit{end-1}.
  
  for i in range(1,21):
      print i

- Range can also take a \textit{stride} value
  
  for i in range(0,21,2):
      print i
Exiting Loops

- It is often necessary to exit from a loop before the condition has been completed. We use the break statement for this.

```python
while x>0:
    if x>1000:
        break
    break
x=x+1
```
Cycling Through Loops

• We also sometimes need to skip an iteration of the loop. The continue statement accomplishes this.

```python
while x>0:
    if x>=10000: break
    if x<100: continue
    x=x+20.
```
The Do-Nothing Statement

- Occasionally we need a statement that does nothing (a no-op). In Python that is the pass statement.

- An infinite loop
  
  while True: pass
Functions

• Functions are reusable blocks of code that have a well-defined purpose, with prescribed inputs and outputs.

• Like most computer languages, Python has a set of built-in functions, such as `len()`

• Users write new functions using the `def` statement.
Functions (Continued)

• Syntax
  def funname(arg1, arg2, arg3):
    code
    return value

• Example (we will discuss the import statement later)
  def enorm(x,y):
    import math
    return math.sqrt(x**2 + y**2)
Variable Passing and Returning

- Python passes variables to functions by assignment, which is like passing by reference (pointers). If the variable is changed in the function it will change in the calling program.

- However, primitive types (integer, long integer, float, complex, boolean) as well as strings and a few others are immutable and will not be changed even if the function changes them; if they are passed a local copy will be made.

- Python functions can accept and return lists and dictionaries.

- Python functions can return multiple values if they are grouped into a tuple. This is one of the major uses of tuples.
Optional and Keyword Arguments

• An argument may be assigned a default value in the parameter list; if so that argument becomes optional. If not present in the calling list it takes the assigned default value.

def func(x,y=0,w=3)

• Keyword arguments can be passed by keyword, not position. They must follow any positional arguments in the argument list.

def func(x,y,w)

z=func(x,w=6,y=2)
Variable Scope

• The *scope* of a variable is the range over which it has a defined value. In Python, the scope of a variable is the code block within which it is first referenced. So a calling program may have a variable named x, and a function may also have a variable named x, and if x is not an argument to the function then it will be distinct from the x in the main program.

• To override the local scope, use the keyword global. This is the only variable declaration statement in Python.

```python
def example(z):
    global x
    return y=z*x
```
Lambda (Anonymous) Functions

• The lambda statement is a way to define a function that is not given a name. It must be possible to express the function in a single line.

• Syntax
  
  `lambda var1, var2: expression`

• Example
  
  `z=lambda x,y: x**y`

  Reference this anonymous function by e.g.
  
  `print z(5,4)`
Exercise

• Type
  
z=lambda x,y:x**2-y

print z(4,5)
Input and Output

- The print statement prints to standard output (usually the console) unless redirected.

  ```
  print var1, var2, var3, list1
  ```

- If the output is to be formatted into an appearance controlled by the programmer, use the `%` operator and conversion characters. The most commonly used are:
  - `d,i` : decimal integer
  - `E, e` : exponential notation (upper- or lower-case E)
  - `F, f` : floating-point number expressed as a decimal
  - `G, g` : general (selects f/F or e/E as appropriate)
  - `s` : string
Further control over appearance is achieved by format specifiers.

- 0 : zero pad
- - : left-justify in the field
- + : include sign even if + (- is always included)
- Integer indicating width of the field
- Period (.) followed by the number of decimal digits to print.
Format Examples

print “The results are %d %f ” % (i, x)
print “The file number is %s.%04i” % (fn, j)
print “Output value: %8.4f” % x

• The % between the format string and the variables is required.

• Note that in this form of print, multiple values must be in a tuple.
Files

- Files are objects that can be manipulated through a set of Python built-in functions.
- First the file should be opened. This associates a file object with it.
- `f=open('name')` # open for reading
- `f=open('name','r')` # open read-only
- `f=open('name','w')` # open for writing
- `f=open('name','rb')` # for binary data
- `f=open('name',wb')` # for binary data
Print to File

- To print to a file, open the file and redirect print's output to it.

```python
f=open('name','w')
print >>f, '%s.%02i' % (fn,j)
```
Reading and Writing to Files

- Once we have opened a file, we can invoke functions on the file
- `f.read([n])`
  - Read at most n bytes (the argument is optional, entire file is read if it is not present)
- `f.readline([n])`
  - Read a single line up to n bytes (entire line if n is omitted)
- `f.write(s)`
  - Write string s – note that write can only write strings. Similarly, read only returns strings.
Other File Functions

- `f.close`
- `f.seek(offset [, where])`
- `f.flush`
  - Handy for buffered input/output
- `f.fileno`
  - Returns an integer
- `f.next`
  - Reads the next line
Modules

- A module is a file that contains groups of related functions and variables or objects.
- Modules permit code reuse and program structure.
- Modules are attached to a caller by the import statement. By default this imports all the symbols but they retain their own namespace. Subsequently they are referenced by modulename.membername.
- The module file must be named modulename.py, but the .py is not included in importation and reloading of the module.
Import

- To use a module, import it
  
  ```python
  import math
  import os
  ```

- Then reference it with the name of the module followed by the name of the member.
  
  ```python
  z=math.sqrt(x**2+y**2)
  homedir=os.getenv("HOME")
  ```

- Imports happen only the first time the statement is encountered. If the module must be reloaded to make changes, use the reload function

  ```python
  reload(modulename)
  ```
Import Variations

- **from** imports only the specified names from the module. Subsequently they do not need a fully qualified name as the reference.

  ```python
  from math import sqrt, cos
  z=sqrt(x**2+y**2)
  ```

- **from * imports all symbols except those beginning with an underscore (\_)**

  ```python
  from math import  *
  z=sqrt(x**2+y**2)
  ```

- Members can be renamed

  ```python
  from math import sqrt as squareroot
  z=squareroot(x**2+y**2)
  ```
Exercise

• Type

    import os
    from math import *
    x=10; y=11
    z=sqrt(x**2+y**2)
    print environ['HOME']
    print os.environ['HOME']
Module Structure

- Variables declared at the top of the file and not included in functions are global to the module.
- All data is accessible, even names beginning with underscores, but it is bad form to try to access lower-level names in the module.
- Python provides a number of built-in attributes that begin and end with double underscores. For example:
  - The top-level code is actually a module called __main__
  - A module can be run as a standalone program if it makes use of the __name__ attribute; specifically,
    - If __name__ == '__main__': do stuff
Classes and OOP

- Python supports object-oriented programming (but does not require it). Classes implement objects defined by the programmer.
- Classes may contain data and functions (called “methods” in OOP parlance).
- When a class is called, a new instance of the object is created.
- Classes *inherit* methods and variables from their parent classes. Multiple inheritance (inheritance from more than one parent class) is supported but should be used judiciously.
class MyClass(object) :
    
    """This class is my class"""

    x=42
    def __init__(self, y):
        self.y=21
    def __del__(self):
        self.y=0
    def myfunc(self,z):
        """Do something"""
        return MyClass.x*self.y-z
Explanation

- Line one defines the class name. The object class is the parent of all classes. (In older Python it was not used, but in new code it should be present.)

- The next line is a documentation string.

- Next is a variable that is global to the class. It is initialized to a value.

- Next is the `__init__` method, also called a constructor. It is executed when a new class instance is created.

- The first parameter to every class method **must** be a placeholder for the class, which by convention (it is not a reserved word) is always called `self`.

- We have a del or destructor, which is executed when the instance is deleted.

- Finally, we have a function that performs a computation.
Using MyClass

- Save MyClass in a file such as mymod.py. Run the interpreter (>>> is its prompt):

```python
>>> from mymod import *
>>> mc = MyClass(12)
>>> mc.x
42
>>> x = 12
>>> x
12
>>> mc.myfunc(19)
863
```
Exercise

• In IDLE, from the File menu open a new window. Type in MyClass as on the earlier slide. Save as mymodule.py

• In your interpreter window, type

```python
from mymodule import *
mc=MyClass(12)
x=12
print x
mc.x
mc.myfunc(19)
```
Pickling

In order to print out objects or read them back from a file, an operation called “pickling” must be performed. This converts the object into a linear stream of bytes to be written or read.

```python
import pickle
aclass=MyClass(11)
f=open('myresults.dat','w')
pickle.dump(aclass,f)
```

To restore use

```python
aclass=pickle.load(f)
```
NumPy

• NumPy is not yet part of Python, but on Linux is generally distributed with it, and it is easy to acquire on other platforms. Its homepage is www.numpy.org

• NumPy adds many features important or useful to scientific/numeric computing. These include
  • True multidimensional arrays
  • Linear algebra functions
  • FFT functions
  • Random number generators
  • Tools for integrating Fortran, C, and C++ libraries
NumPy Arrays

- NumPy Arrays have a format similar to Fortran, Matlab, and R, but unlike those languages their indices start at 0 and go to m-1.
  - from numpy import *
  - Arrays are objects and instances must be created with the array function, or with the zeros, ones, or identity function.
    - A=array([[0,1],[2,2]])
    - B=zeros((2,3))
    - C=ones((4,4))
    - I=identity(5)
  - Arithmetic operations on arrays take place elementwise (like Fortran, or like Matlab's "dot" operators).
    - A+3
Array Elements and Slices

- Array elements can be accessed by enclosing the indices as a comma-separated list enclosed in square brackets.
  - \( A = \text{identity}(2) \)
  - \( A[0,0] \) is 1
- Subarrays can be defined similarly to list slices
  - \( A[0,:) \) is the first row
  - \( A[:,1] \) is the second column
  - Note: bounds can be included and follows the same convention as for list and string slices, i.e. it goes to \( \text{ub}-1 \)
Ufuncs (Universal Functions)

- NumPy overloads several math-oriented Python functions to operate on arrays. They are applied elementwise. These include the arithmetic operators (+, *, -, /), %, **, trigonometric functions (including inverse functions), hyperbolic trig functions, exp, log, log10, and a few others. See NumPy documentation for details.
Array Functions

- Several array functions have been added, among them
  - transpose
  - reshape
  - where
  - diagonal
  - trace
  - dot
  - matrixmultiply
- And many others
SciPy

- SciPy provides additional scientific packages on top of NumPy. These include functions for
  - Special functions
  - Optimizations
  - Linear algebra
  - Quadrature
  - Interpolation
  - Signal processing
  - Statistics

- See www.scipy.org for details.
Importing SciPy Packages

• Packages must be imported by name; from * will not suffice.
  from scipy import *
  from scipy import linalg, optimize

• For a loaded package, type
  help(linalg)
  for details.
SciPy Packages

• The packages are
  • cluster
  • constants
  • fftpack
  • integrate
  • interpolate
  • io
  • linalg
  • maxentropy
  • ndimage

  • odr
  • optimize
  • signal
  • sparse
  • spatial
  • special
  • stats
  • weave
Matplotlib

- Matplotlib is another Python package that can produce publication-quality plots similar to those of Matlab. Its homepage is at matplotlib.sourceforge.net
- A full complement of plot types is available.
  - Line plots
  - Scatter plots
  - Histograms
  - Bar charts
  - Pie charts
  - Contour plots
Simple Matplotlib Example

```python
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(10*np.random.randn(100), 10*np.random.randn(100), 'o')
ax.set_title('Scatter Plot')
plt.show()
```
Result of this Example
Some References

- Mark Lutz and David Ascher, Learning Python (O'Reilly)
  Third edition available on Safari via Virgo
- Mark Lutz, Programming Python (O'Reilly)
  Third edition available on Safari via Virgo
- Michael Dawson, Python Programming for the Absolute Beginner
  Electronic resource on Virgo