A Brief Schematic of Python

Prof Wm C Bauldry

MAT 4310
Spring, 2013

Outline

1. Launching IDLE 2
   - a. Mac OS X / Windows
   - b. Portable Python (Win)

2. Operators 3
   1. Arithmetic Operators
   2. Comparison Operators

3. Variables 5
   1. Variables & Assignment
   2. Variable Names & Reserved Words

4. Flow Control 7
   1. Conditional Statements
   2. Loops

5. Functions 9
   1. Defining Functions
   2. Function Arguments
   3. Standard Math Functions

6. Code 13
   1. Pseudocode $\Longrightarrow$ Python
   2. Coding Style

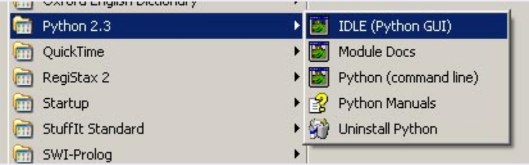
7. Lists .16
   - a. Defining Lists
   - b. Indexing
   - c. Functions and Methods

Launching IDLE

Starting Python in IDLE

Mac OS X: Double click the "Run Python IDLE" app OR
Execute "idle &" in a Terminal window (on ASU Macs).

Windows: Choose "IDLE" from the submenu of Python in the Start Menu (see below).



- To run a Python program or script in IDLE, choose the menu item RUN ▶RUN MODULE after opening the program file.
- On ASU public PCs, it's easiest to use Portable Python on a flash drive. (*First launch Python, then start IDLE; reverse to quit.*)

Python Arithmetic Operators

Standard Arithmetic Operators in Python

Operator	Description	Example
$+$	Addition	$2+3 \to 5$
$-$	Subtraction	$2-3 \to -1$
$*$	Multiplication	$2*3 \to 6$
$/$	Division	$2/3 \to 0$ $2.0/3.0 \to 0.666\ldots$
$\%$	Modulus	$2\%3 \to 2$
$**$	Exponent	$2**3 \to 8$
$//$	Floor Division	$2//3 \to 0$ $2.0//3.0 \to 0.0$

Python Comparison Operators

Standard Comparison Operators in Python

Operator	Description
==	Equal
!=	Not equal
<>	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Python Assignment Operators

Identifiers

Variable: A name begins with a letter A to z or an underscore, possibly followed by letters, numbers, or underscores. Standard variables begin with lower case; class names begin with capitals.

Assignment Operators in Python

Operator	Type	Description
$=$	Simple assignment	$c = a + b$ will assign $a + b$ into c
$+=$	Add and assign	$c += a \iff c = c + a$
$-=$	Subtract and assign	$c -= a \iff c = c - a$
$*=$	Multiply and assign	$c *= a \iff c = c * a$
$/=$	Divide and assign	$c /= a \iff c = c / a$
$\%=$	Modulus and assign	$c \%= a \iff c = c \% a$
$**=$	Exponentiate and assign	$c **= a \iff c = c ** a$
$//=$	Floor Divide and assign	$c //= a \iff c = c // a$

Python's Reserved Words

Reserved Words

The following *reserved words* may not be used as identifiers.

and	assert	break	class	continue
def	del	elif	else	except
exec	finally	for	from	global
if	import	in	is	lambda
not	or	pass	print	raise
return	try	while	with	yield

Also, do not use predefined function names for variables.

Data	Float	Int	Numeric	Oxphys
array	close	float	int	input
open	range	type	write	zeros

Python Conditional Statements

If – Then – Else

Simple If:

```
if condition:
    statements...
```

Compund If:

```
if condition:
    statements...
else:
    statements...
```

```
if condition:
    statements...
elif condition:
    statements...
else:
    statements...
```

Notes:

- Required colons end the `if`, `elif`, and `else` lines.
- Statements are made into *suites* (blocks of statements) by indentation.
- A different indentation level terminates a suite.
- Several `if`s may be nested.
- The `else` clauses are optional.

Python Loops

While Loop

While: `while` *condition*:
    statements. . .

```
while x <= 10:
    x += 1
```

For: `for` *var_name* in *list*:
    statements. . .

```
for i in range(1,10):
    y[i] = -i
```

The syntax of `range` is

`range(`<*start*,> *stop* <, *step*>`)`

E.g.[1],

```
>>> range(4)
  [0, 1, 2, 3]
>>> range(1,7,2)
  [1, 3, 5]
```

```
>>> y = range(4)
>>> for i in range(3):
        y[i] = -i

>>> y
  [0, -1, -2, 3]
```

[1] See also Tutorial Point's examples.

Defining Functions in Python

Functions

Function:

```
def fcn_name(parameters):
    "description" # optional, description string
    function_suite
    return <value(s)> # optional, returned values
```

E.g.:

```
>>> def eratosthenes(n):
      "Prime sieve"
      if n>1:  print 2, ’is prime’
      for num in range(2,n+1):
        for i in range(2,num):
          if num%i==0:
            break
          elif i==num-1:
            print num, ’is prime’
>>> eratosthenes(5)
  2 is prime
  3 is prime
  5 is prime
```

Function Arguments, I

Defining a Function's Arguments

Required arguments: given by a sequence of valid name(s) in the function definition:

`def f(x):` or `def f(x,y):`

Default arguments: values given by equations in the function definition and are optional in the calls:

`def f(x,y=30):` $\Longrightarrow$ `f(1,2)` or `f(1)` are valid

`def f(x=1,y):` is **not** valid (default args must come last)

- The order of arguments defined is static.
- Arguments can be used as "keywords" in any order: `def f(x,y):` can be called with `f(y=2,x=1)`
- Variables not specified as arguments are local to the function
- A variable number of arguments can be indicated with an asterisk via: `def f(x,*`*name*`):`

Function Arguments, II

How are arguments passed? By ‘reference-to-object by value.’

- Strings, numbers, and tuples are *immutable objects:* Altering them inside a function creates a new instance; the original object **is not changed**.
- Lists and dictionaries are *mutable objects* (you can change the object in-place): Altering them inside a function creates a new instance; however, the original object **is changed**.

```
>>> a = 1
>>> def f(x):
       x = 2
       return 0

>>> f(a)
 0
>>> a
 1
```

```
>>> a = [0,1,2,3]
>>> def f(x):
       x.append("new")
       return 0

>>> f(a)
 0
>>> a
 [0, 1, 2, 3, ’new’]
```

Standard Mathematical Functions

Accessing Math Functions in Python

1. Load the math module[2] (*Enter* `help(math)` *or* `help(math.`*fcn*`)` *for help*):

   ```
   >>> import math
   ```

2. Use the construct `math.`*fcn*`(`*args*`)`:

   ```
   >>> math.sin(math.pi)
     1.2246467991473532e-16
   ```

The standard functions are:

math.pi	math.e	math.ceil(x)	math.fabs(x)
math.factorial(n)	math.floor(x)	math.fmod(x,y)	math.modf(x,y)
math.trunc(x)	math.exp(x)	math.log(x[, base])	math.log10(x)
math.pow(x,y)	math.sqrt(x)	math.cos(x)	math.sin(x)
math.tan(x)	math.acos(x)	math.asin(x)	math.atan(x)
math.degrees(x)	math.radians(x)	math.hypot(x,y)	math.cosh(x)
math.sinh(x)	math.tanh(x)	math.acosh(x)	math.asinh(x)
math.atanh(x)	math.erf(x)	math.erfc(x)	math.gamma(x)

For complex values, use the `cmath` module. (*Import, then see* `help(cmath)`.)

[2] Alternate: Use “`from math import *`” Then use `sin` instead of `math.sin`, &c.

Pseudocode $\Longrightarrow$ Python

Program First (*Cheney & Kincaid,* pg 10)

```
program First

  integer $i, imax, n \leftarrow 30$
  real $err, y, x \leftarrow 0.5, h \leftarrow 1, emax \leftarrow 0$
  for $i = 1$ to $n$ do
     $h \leftarrow 0.25h$
     $y \leftarrow [\sin(x+h) - \sin(x)]/h$

     $err \leftarrow |\cos(x) - y|$
     output $i, h, y, err$
     if $err > emax$ then
        $emax \leftarrow err$; $imax \leftarrow i$
     end if
  end for
  output $imax, emax$
  end program First
```

```
>>> def first():
    import math
    n=30
    x,h,emax=0.5,1.0,0.0
    for i in range(n):
      h=0.25*h    [or h*=0.25]
      y=(math.sin(x+h)
          -math.sin(x))/h
      err=abs(math.cos(x)-y)
      print i,h,y,err
      if err>emax:
        emax=err
        imax=i
      print imax,emax
```

Pseudocode ⟹ Python, II

Program First v 2

```
>>> from math import *
>>> def first_v2(n):
   x,h,emax = 0.5, 1.0, 0.0
   sinx,cosx = sin(x), cos(x)
   for i in range(n):
     h *= 0.25
     y = (sin(x+h) - sinx)/ h
     err = abs(cosx - y)
     print i,h,y,err
     if err > emax:
       emax,imax = err,i
   print (imax, emax)

>>> first_v2(30)
0 0.25 0.808852885677 0.0687296762138
1 0.0625 0.862034158909 0.0155484029813
     ⋮
(26, 0.8775825618903728)
```

Coding Style

Coding Style Guides[3]

- Use 4-space indentation, no tabs.
- Wrap lines so that they don't exceed 79 characters.
- Use blank lines to separate functions and classes, and larger blocks of code inside functions.
- Put comments on a line of their own.
- Use *docstrings* in function definitions.
- Use spaces around operators and after commas, but not directly inside bracketing constructs: `a = f(1,2) + g(3,4)`.
- Name your functions consistently; the convention is to use lower_case_with_underscores.
- Don't use fancy encodings; plain ASCII work best.

[3] From the *PEP 8 – Style Guide for Python Code*.

Lists in Python

Python Lists

List: an ordered set of objects inside brackets.

```
>>> L = [1,3,"s",1.1,0.]
>>> L
  [1, 3, ’s’, 1.1, 0.0]
```

```
>>> range(2,9,2)
 [2, 4, 6, 8]
```

- Indexing origin is 0. Then `L[j]` gives the $(j+1)$st element.
- The last element is `L[len(L)-1]` or `L[-1]`; 2nd last is `L[-2]`, &c.
- `L1 + L2` returns the concatenation of `L1` and `L2`
- `n * L` returns the concatenation of n copies of `L`
- Apply a function to a list's elements with `map(f,L)`
- Syntax for *slices* (similar to `range`): L[<*start*>:<*stop*><:*step*>]

```
>>> L = range(10)
>>> L[:4]
  [0, 1, 2, 3]
>>> L[3::2]
  [3, 5, 7, 9]
```

```
>>> L[-3::2]
  [7, 9]
>>> L[-4::-1]
  [6, 5, 4, 3, 2, 1, 0]
```

List Functions & Methods in Python, I

List Functions

Assume a list object named `theList`. A list **must be defined** before using it.

1. `len`(object) → integer ⇐ Number of items.
2. `max`(list) → value ⇐ Largest item.
3. `min`(list) → value ⇐ Smallest item.
4. `any`(tuple) → boolean ⇐ True if <u>any</u> item is True.
5. `all`(tuple) → boolean ⇐ True if <u>all</u> items are True.

List Functions & Methods in Python, I

List Functions

Assume a list object named `theList`. A list **must be defined** before using it.

1. `len`(object) → integer ⇐ Number of items.
2. `max`(list) → value ⇐ Largest item.
3. `min`(list) → value ⇐ Smallest item.
4. `any`(tuple) → boolean ⇐ True if any item is True.
5. `all`(tuple) → boolean ⇐ True if all items are True.

List Information Methods

Assume a list object named `theList`.

1. `theList.count`(value) → integer ⇐ Number of occurrences of value.
2. `theList.index`(value) → integer ⇐ Index of first occurrence of value.

List Functions & Methods in Python, II

List Updating Methods

Assume a list object named `theList`.

1. `theList.append`(object) ⇐ Append object to end of the list.

List Functions & Methods in Python, II

List Updating Methods

Assume a list object named `theList`.

1. `theList.append`(object) ⇐ Append object to end of the list.

2. `theList.extend`(list) ⇐ Extend by appending list elements; different from append(object) which treats the argument as a single object.

List Functions & Methods in Python, II

List Updating Methods

Assume a list object named `theList`.

1. `theList.append`(object) ⇐ Append object to end of the list.
2. `theList.extend`(list) ⇐ Extend by appending list elements; different from append(object) which treats the argument as a single object.
3. `theList.insert`(index,object) ⇐ Insert object before position index. If index $>$ `len`(list), object is appended; if index < 0, object is prepended.

List Functions & Methods in Python, II

List Updating Methods

Assume a list object named `theList`.

1. `theList.append`(object) ⇐ Append object to end of the list.
2. `theList.extend`(list) ⇐ Extend by appending list elements; different from append(object) which treats the argument as a single object.
3. `theList.insert`(index,object) ⇐ Insert object before position index. If index $>$ `len`(list), object is appended; if index < 0, object is prepended.
4. `theList.pop`([index]) → item ⇐ Remove and return item at index (default: last, -1). *Exception is raised if the list is empty.*

List Functions & Methods in Python, II

List Updating Methods

Assume a list object named `theList`.

1. `theList.append`(object) ⇐ Append object to end of the list.

2. `theList.extend`(list) ⇐ Extend by appending list elements; different from append(object) which treats the argument as a single object.

3. `theList.insert`(index,object) ⇐ Insert object before position index. If index $>$ `len`(list), object is appended; if index < 0, object is prepended.

4. `theList.pop`([index]) → item ⇐ Remove and return item at index (default: last, -1). *Exception is raised if the list is empty.*

5. `theList.remove`(value) → item ⇐ Remove first occurrence of value. *Exception is raised if value is not in the list.*

List Functions & Methods in Python, II

List Updating Methods

Assume a list object named theList.

1. theList.append(object) $\Leftarrow$ Append object to end of the list.
2. theList.extend(list) $\Leftarrow$ Extend by appending list elements; different from append(object) which treats the argument as a single object.
3. theList.insert(index,object) $\Leftarrow$ Insert object before position index. If index $>$ len(list), object is appended; if index < 0, object is prepended.
4. theList.pop([index]) $\rightarrow$ item $\Leftarrow$ Remove and return item at index (default: last, -1). *Exception is raised if the list is empty.*
5. theList.remove(value) $\rightarrow$ item $\Leftarrow$ Remove first occurrence of value. *Exception is raised if value is not in the list.*
6. theList.reverse $\Leftarrow$ Reverse "in place," does not create a new list.

List Functions & Methods in Python, II

List Updating Methods

Assume a list object named `theList`.

1. `theList.append`(object) ⇐ Append object to end of the list.
2. `theList.extend`(list) ⇐ Extend by appending list elements; different from append(object) which treats the argument as a single object.
3. `theList.insert`(index,object) ⇐ Insert object before position index. If index $>$ `len`(list), object is appended; if index < 0, object is prepended.
4. `theList.pop`([index]) → item ⇐ Remove and return item at index (default: last, -1). *Exception is raised if the list is empty.*
5. `theList.remove`(value) → item ⇐ Remove first occurrence of value. *Exception is raised if value is not in the list.*
6. `theList.reverse` ⇐ Reverse "in place," does not create a new list.
7. `theList.sort`([cmpfunc]) ⇐ Sort "in place," does not create a new list. If a comparison function, cmpfunc is given, it must behave like the built-in `cmp`: cmpfunc(x,y) $\rightarrow -1, 0, 1$.