New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Advanced Python

Executive Summary, Session 1

OBJECT: a ***unit of data*** of a particular ***type*** with characteristic ***functionality*** (i.e., methods and/or use with operators). Everything in Python is an object.

- "atomic" data: **int**eger, **float**, **str**ing, **bool**ean, **None**
- "container" data: **list**, **tuple**, **set**, **dict**
- "code" objects: functions, methods, classes
- "custom" objects: defined by Python module authors (including you)

VARIABLE: an object *bound* (assigned to) a name.

```
var = 10
myxx = "hello!"

def myfunc():
    print 'OK!'
```

Initialization of a variable means that the object is being stored in memory for later use during the run of the program. ***Re-initialization*** means that the name has been bound to a new object, and is now unbound from the prior object. All initializations create a new binding between a name and an object.

STATEMENT

simple statements:

• assignment (with =):	`var = 10`
• augmented assignment (+=, -=, etc.):	`var += 5`
• **del** (unbind a variable or remove a dict key/value):	`del var`
• **print** (echo string text to **STDOUT**):	`print 'hello!'`
• **break** (drop out of a loop):	`break`
• **continue** (jump to next iteration of a loop):	`continue`
• **import** (import a Python module):	`import random`

compound statements:

• **if**, **elif**, **else**:	`if var > 5:`
• **and**, **or** (for compound tests):	`if var > 5 and var < 10:`
• **not** (testing negative condition):	`if not var > 5:`
• **while** loop:	`while var < 100:`
• **for**: iterate through an iterable (container or file):	`for item in mylist:`
• **try**: test code for a raised exception:	`try:`
• **except**: lines to execute if exception occurs	`except IndexError:`
• **def**: declare a function:	`def myfunc(arg1, arg2):`

OPERATORS

An *operator* usually has two *operands* upon which it operates; it returns the result.

• math operators (**+**, **-**, *, **/**, **, **%**):	`var = 5 * 10`
• binary math operators (**+=**, **-=**, ***=**, **/=**, ****=**):	`var += 1`
• comparison operators (**<**, **>**, **==**, **<=**, **>=**, **!=**):	`if var < 20:  print var`
• membership operators (**in**, **not in**):	`if 'David' not in names_list:`
• identity operators (**is**, **is not**):	`if var is None:`
• boolean compound operators (**and**, **or**):	`if var > 10 and var < 20:`
• ternary expression (**x if C else y**):	`var = 0 if var < 0 else var`

FUNCTIONS

• **len()**: length of a string or container	`mylen = len('hello')`
• **round()**: round a float	`myr = round(5.539, 2)`
• **type()**: get type of any object	`print type(myr)`
• **raw_input()**: take keyboard input	`x = raw_input('enter num: ')`
• **exit()**: exit the program	`exit()`
• **int()**, **float()**, **str()**, **bool()**: object constructors	`myf = float(5)`
• **repr()**: display an object in more 'literal' form	`print repr(mystr)`
• **any()**: given a container, **True** if any are **True**	`if any([5, 0, 0, 0.0, None]):`
• **all()**: given a container, **True** if all are **True**	`if all([5, 10, 0.9, True]):`
• **min(), max()**: min or max val from an iterable	`this = min([5, 9, 3, 0.9, 2])`
• **sorted()**: return a list of sorted objects	`x = sorted([5, 9, 3, 0.9, 2])`
• **range()**: return a list of integers in a range	`myrange = range(5, 10)`
• **enumerate()**: return a list of (count, item)	`for count, item in enumerate(mylst):`

CORE OBJECT TYPES

INTEGER and **FLOAT**

```
var = 5                 # initialize an int object
my_xx = 5.0             # initialize a float object
```

STRING

```
mystr = 'hello'         # initialize a single-quoted string object
yourzzy = "hello"       # initialize a double-quoted string (same as above)
this_one = """this      # initialize a multi-line string object
is a
multi-line
string."""
```

• **string slicing**: return a portion of string	`slicstr = mystr[3:5]`
• **upper(), lower()**: return an upper- or lowercased str	`newstr = mystr.upper()`
• **count()**: return int occurrences of substr within a string	`mynum = mystr.count('e')`
• **find()**: return int index position of substr within a string	`indexx = mystr.index('e')`
• **replace()**: return a new string with substr replaced	`rstr = mystr.replace('e', 'a')`
• **format()**: return a new string with **{}** tokens replaced	`fmstr = mystr.format(55, 23.0)`
• **isdigit()**: return **True** if string is all digit characters	`if mystr.isdigit():`
• **startswith()**: return **True** if string begins with substr	`if mystr.startswith('hel'):`
• **split()**: return a list of strs from string split on delimeters	`slist = mystr.split()`
• **splitlines()**: return a list of strs comprising lines from file	`slines = text.splitlines()`
• **rstrip()**: return a str with whitespace / other chars removed	`mystr = mystr.rstrip()`

LIST and **TUPLE**

• **subscripting**: return one item from a list	`myitem = mylist[3]`
• **slicing**: return a list or tuple withs elected items	`newlist = mylist[3:5]`
• list **append()**: add an item to the end of a list	`mylist.append(100)`

SET

• **add()**: add an item to a set	`myset.add('newitem')`

FILE

```
fh = open('thisfile.txt')                  # initialize a file object
```

• **'for' looping**: assign each line in file to a 'control variable'	`for line in fh:`
• **read()**: return a string containing the text of the file	`text = fh.read()`
• **readlines()**: return a list of strings, each line from file	`lines = fh.readlines()`

CONTAINER OPERATIONS (applies to **list**, **tuple**, **set**, **str**ing, others)

• **len()**: get int length of (# of items in) container	`length = len(mylist)`
• **in**: test for membership in a container	`if 'hello' in mylist:`
• **for**: loop through each item in a container	`for item in myset:`
• **sum()**: sum values in a container (numbers)	`total = sum(prices)`
• **max()**: get max value in a container	`highest = max(prices)`
• **min()**: get min value in a container	`minim = min(prices)`
• **sorted()**: return list of items, sorted	`slist = sorted(mylist)`
• **subscript**: return item at index pos (list, tuple)	`first = mylist[0]`
• **slice**: return new container with selected items	`newlist = mylist[3:5]`

EXCEPTIONS (also see Exceptions Reference)

- **SyntaxError**: when the code has a syntax mistake (missing paren or quote, etc.)
- **NameError**: when a variable name is used that doesn't exist (often a misspelling)
- **TypeError**: when the type of object used in a function or method is incorrect
- **AttributeError**: when attempting to acces an attribute (e.g. method) that is incorrect for the object
- **IndexError**: when attempting to access an item in a list where the index doesn't exist