CPSC 1301, Computer Science I Lab Assignment

Lab 04a

1 Three Key Ideas

The book discusses three key ideas that provide a foundation to Python. You will find these ideas too advanced at this point, but you should at least be introduced to them. In a pharse, we can describe them as follows: (1) *everything in Python is an object*, (2) *Python codes to interfaces rather than types*, and (3) *Python types are either immutable or mutable.*

everything in Python is an object You will get to objects in your next course. We do not learn about objects here. That said, you can think of an object as a *bundle of stuff.* "stuff" is a technical term that means a package of variables and functions. For example, the integer 42 is not just an integer, but a bundle of

> abs, add, and, bool, ceil, class, delattr, dir, divmod, doc, eq, float, floor, floordiv, format, ge, getattribute, getnewargs, gt, hash, index, init, init_subclass, int, invert, le, lshift, lt, mod, mul, ne, neg, new, or, pos, pow, radd, rand, rdivmod, reduce, reduce_ex, repr, rfloordiv, rlshift, rmod, rmul, ror, round, rpow, rrshift, rshift, rsub, rtruediv, rxor, setattr, sizeof, str, sub, subclasshook, truediv, trunc, xor, as_integer_ratio, bit_length, conjugate, denominator, from_bytes, imag, numerator, real, to_bytes

While the string '42' is not just a string, but a bundle of

> add, class, contains, delattr, dir, doc, eq, format, ge, getattribute, getitem, getnewargs, gt, hash, init, init_subclass, iter, le, len, lt, mod, mul, ne, new, reduce, reduce_ex, repr, rmod, rmul, setattr, sizeof, str, subclasshook, capitalize, casefold, center, count, encode, endswith, expandtabs, find, format, format_map, index, isalnum, isalpha, isascii, isdecimal, isdigit, isidentifier, islower, isnumeric, isprintable, isspace, istitle, isupper, join, ljust, lower, lstrip, maketrans, partition, replace, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper, zfill

Python codes to interfaces rather than types In Python, we don't care what an object *is* but what it *does.* We call this *polymorphism.* For example, when we send an object a message `play()`, the behavior differs as to whether the object is a flute, guitar, drum, football, golf ball, pack of cards, or radio. We only care about the behavior of the object, not what it actually is. We sometimes call this *duck typing.* That is, if it quacks like a duck, waddles like a duck, and swims like a duck, it is a duck.

Python types are either immutable or mutable This is perhaps the most important distinction. An *immutable* type cannot be changed but must be reassigned, while a *mutable* type cn be changed in place. I will an example of this in class.

2 Lab

Run the following commands in your Python interpreter. Submit a transcript of your session as your deliverable for this lab. This should be a plain text file named `lab02a lastname.txt`.

```
>>> #Numbers
>>> 123 + 222 # Integer addition
>>> 1.5 * 4 # Floating-point multiplication
>>> 2 ** 100 # 2 to the power 100, again
>>> len(str(2 ** 1000000)) # How many digits in a really BIG number?
>>> 3.1415 * 2 # repr: as code (Pythons < 2.7 and 3.1)
>>> print(3.1415 * 2) # str: user-friendly
>>> import math
>>> math.pi
>>> math.sqrt(85)
>>> import random
>>> random.random()
>>> random.choice([1, 2, 3, 4])

>>> #Strings

>>> #Create an object
>>> 'spam'
>>> S = 'Spam' # Make a 4-character string, and assign it to a name
>>> len(S) # Length
>>> S[0] # The first item in S, indexing by zero-based position
>>> S[1] # The second item from the left
>>> S[-1] # The last item from the end in S
>>> S[-2] # The second-to-last item from the end
>>> S[-1] # The last item in S
>>> S[len(S)-1] # Negative indexing, the hard way
>>> S # A 4-character string
>>> S[1:3] # Slice of S from offsets 1 through 2 (not 3)
>>> S[1:] # Everything past the first (1:len(S))
>>> S # S itself hasn't changed
>>> S[0:3] # Everything but the last
>>> S[:3] # Same as S[0:3]
>>> S[:-1] # Everything but the last again, but simpler (0:-1)
>>> S[:] # All of S as a top-level copy (0:len(S))
>>> S
>>> S + 'xyz' # Concatenation
>>> S # S is unchanged
>>> S * 8 # Repetition

>>> #Immutability, strings cnnot be changed
>>> S
>>> S[0] = 'z' # Immutable objects cannot be changed
>>> S = 'z' + S[1:] # But we can run expressions to make new objects
>>> S
>>> S = 'shrubbery'
>>> L = list(S) # Expand to a list: [...]
>>> L
>>> L[1] = 'c' # Change it in place
>>> ''.join(L) # Join with empty delimiter
>>> S = 'Spam'
>>> S.find('pa') # Find the offset of a substring in S
>>> S
>>> S.replace('pa', 'XYZ') # Replace occurrences of a string in S with another
>>> S
 q>>> line = ’aaa,bbb,ccccc,dd’
>>> line.split(’,’) # Split on a delimiter into a list of substrings
>>> S = ’spam’
>>> S.upper() # Upper- and lowercase conversions
>>> S.isalpha() # Content tests: isalpha, isdigit, etc.
>>> line = ’aaa,bbb,ccccc,dd\\n’
>>> line.rstrip() # Remove whitespace characters on the right side
>>> line.rstrip().split(’,’) # Combine two operations
>>> ’%s, eggs, and %s’ % (’spam’, ’SPAM!’) # Formatting expression (all)
>>> ’{0}, eggs, and {1}’.format(’spam’, ’SPAM!’) # Formatting method (2.6+, 3.0+)
>>> ’{}, eggs, and {}’.format(’spam’, ’SPAM!’) # Numbers optional (2.7+, 3.1+)
>>> ’{:,.2f}’.format(296999.2567) # Separators, decimal digits
>>> ’%.2f | %+05d’ % (3.14159, -42) # Digits, padding, signs

>>> #Getting help
>>> dir(S)
>>> S + ’NI!’
>>> S.\_\_add\_\_(’NI!’)
>>> help(S.replace)

>>> #Omit Unicode strings

>>> #Pattern matching
>>> import re
>>> match = re.match(’Hello[ \t]*(.*)world’, ’Hello Python world’)
>>> match.group(1)
>>> match = re.match(’[/:](.*)[/:](.*)[/:](.*)’, ’/usr/home:lumberjack’)
>>> match.groups()
>>> re.split(’[/:]’, ’/usr/home/lumberjack’)

>>> #Lists
>>> L = [123, ’spam’, 1.23] # A list of three different-type objects
>>> len(L) # Number of items in the list
>>> L[0] # Indexing by position
>>> L[:-1] # Slicing a list returns a new list
>>> L + [4, 5, 6] # Concat/repeat make new lists too
>>> L * 2
>>> L # We’re not changing the original list
>>> L.append(’NI’) # Growing: add object at end of list
>>> L
>>> L.pop(2) # Shrinking: delete an item in the middle
>>> L # "del L[2]" deletes from a list too
>>> M = [’bb’, ’aa’, ’cc’]
>>> M.sort()
>>> M
>>> M.reverse()
>>> M
>>> L
>>> L[99]
>>> L[99] = 1
>>> M = [[1, 2, 3], # A 3 3 matrix, as nested lists
>>> M
>>> M[1] # Get row 2
>>> M[1][2] # Get row 2, then get item 3 within the row
>>> col2 = [row[1] for row in M] # Collect the items in column 2
>>> col2
>>> M # The matrix is unchanged
>>> [row[1] + 1 for row in M] # Add 1 to each item in column 2
>>> [row[1] for row in M if row[1] % 2 == 0] # Filter out odd items
>>> diag = [M[i][i] for i in [0, 1, 2]] # Collect a diagonal from matrix
>>> diag
>>> doubles = [c * 2 for c in ’spam’] # Repeat characters in a string
>>> doubles
>>> list(range(4)) # 0..3 (list() required in 3.X)
>>> list(range(-6, 7, 2)) # -6 to +6 by 2 (need list() in 3.X)
>>> [[x ** 2, x ** 3] for x in range(4)] # Multiple values, "if" filters
>>> [[x, x / 2, x * 2] for x in range(-6, 7, 2) if x > 0]
>>> G = (sum(row) for row in M) # Create a generator of row sums
>>> next(G) # iter(G) not required here
>>> next(G) # Run the iteration protocol next()
>>> next(G)
>>> list(map(sum, M)) # Map sum over items in M
>>> {sum(row) for row in M} # Create a set of row sums
>>> {i : sum(M[i]) for i in range(3)} # Creates key/value table of row sums
>>> [ord(x) for x in ’spaam’] # List of character ordinals
>>> {ord(x) for x in ’spaam’} # Sets remove duplicates
>>> {x: ord(x) for x in ’spaam’} # Dictionary keys are unique
>>> (ord(x) for x in ’spaam’) # Generator of values

>>> #Dictionaries
>>> D = {’food’: ’Spam’, ’quantity’: 4, ’color’: ’pink’}
>>> D[’food’] # Fetch value of key ’food’
>>> D[’quantity’] += 1 # Add 1 to ’quantity’ value
>>> D
>>> D = {}
>>> D[’name’] = ’Bob’ # Create keys by assignment
>>> D[’job’] = ’dev’
>>> D[’age’] = 40
>>> D
>>> print(D[’name’])
>>> bob1 = dict(name=’Bob’, job=’dev’, age=40) # Keywords
>>> bob1
>>> bob2 = dict(zip([’name’, ’job’, ’age’], [’Bob’, ’dev’, 40])) # Zipping
>>> bob2
>>> rec = {’name’: {’first’: ’Bob’, ’last’: ’Smith’}, ’jobs’: [’dev’, ’mgr’], ’age’: 40.5}
>>> rec[’name’] # ’name’ is a nested dictionary
>>> rec[’name’][’last’] # Index the nested dictionary
>>> rec[’jobs’] # ’jobs’ is a nested list
>>> rec[’jobs’][-1] # Index the nested list
>>> rec[’jobs’].append(’janitor’) # Expand Bob’s job description in place
>>> rec
>>> D = {’a’: 1, ’b’: 2, ’c’: 3}
>>> D
>>> D[’e’] = 99 # Assigning new keys grows dictionaries
>>> D
>>> D[’f’] # Referencing a nonexistent key is an error
>>> ’f’ in D
>>> if not ’f’ in D: # Python’s sole selection statement
>>> ... print(’missing’)
>>> if not ’f’ in D:
>>> ... print(’missing’)
>>> ... print(’no, really...’) # Statement blocks are indented
>>> value = D.get(’x’, 0) # Index but with a default
>>> value
>>> value = D[’x’] if ’x’ in D else 0 # if/else expression form
>>> value
>>> D = {’a’: 1, ’b’: 2, ’c’: 3}
>>> D
>>> Ks = list(D.keys()) # Unordered keys list
>>> Ks # A list in 2.X, "view" in 3.X: use list()
>>> Ks.sort() # Sorted keys list
>>> Ks
>>> for key in Ks: # Iterate though sorted keys
>>> D
>>> for key in sorted(D):
>>> ... print(key, ’=>’, D[key])
>>> for c in ’spam’:
>>> ... print(c.upper())
>>> x = 4
>>> while x > 0:
>>> ... print(’spam!’ * x)
>>> ... x -= 1
>>> squares = [x ** 2 for x in [1, 2, 3, 4, 5]]
>>> squares
>>> squares = []
>>> for x in [1, 2, 3, 4, 5]: # This is what a list comprehension does
>>> ... squares.append(x ** 2) # Both run the iteration protocol internally
>>> squares

>>> #Tuples
>>> T = (1, 2, 3, 4) # A 4-item tuple
>>> len(T) # Length
>>> T[0] # Indexing, slicing, and more
>>> T.index(4) # Tuple methods: 4 appears at offset 3
>>> T.count(4) # 4 appears once
>>> T[0] = 2 # Tuples are immutable
>>> T = (2,) + T[1:] # Make a new tuple for a new value
>>> T
>>> T = ’spam’, 3.0, [11, 22, 33]
>>> T[1]
>>> T[2][1]
>>> T.append(4)

>>> #Files
>>> f = open(’data.txt’, ’w’) # Make a new file in output mode (’w’ is write)
>>> f.write(’Hello\\n’) # Write strings of characters to it
>>> f.write(’world\\n’) # Return number of items written in Python 3.X
>>> f.close()
>>> f = open(’data.txt’) # ’r’ (read) is the default processing mode
>>> text = f.read() # Read entire file into a string
>>> text
>>> print(text) # print interprets control characters
>>> text.split() # File content is always a string
>>> for line in open('data.txt'): print(line)

>>> #Skip binary files

>>> #Skip Unicode files

>>> #Sets
>>> X = set('spam') # Make a set out of a sequence in 2.X and 3.X
>>> Y = {'h', 'a', 'm'} # Make a set with set literals in 3.X and 2.7
>>> X, Y # A tuple of two sets without parentheses
>>> X \& Y # Intersection
>>> X | Y # Union
>>> X - Y # Difference
>>> X > Y # Superset
>>> {n ** 2 for n in [1, 2, 3, 4]} # Set comprehensions in 3.X and 2.7
>>> list(set([1, 2, 1, 3, 1])) # Filtering out duplicates (possibly reordered)
>>> set('spam') - set('ham') # Finding differences in collections
>>> set('spam') == set('asmp') # Order-neutral equality tests (== is False)
>>> 'p' in set('spam

>>> #Miscellaneous
>>> 1 / 3 # Floating-point (add a .0 in Python 2.X)
>>> (2/3) + (1/2)
>>> import decimal # Decimals: fixed precision
>>> d = decimal.Decimal('3.141')
>>> d + 1
>>> decimal.getcontext().prec = 2
>>> decimal.Decimal('1.00') / decimal.Decimal('3.00')
>>> from fractions import Fraction # Fractions: numerator+denominator
>>> f = Fraction(2, 3)
>>> f + 1
>>> f + Fraction(1, 2)
>>> 1 > 2, 1 < 2 # Booleans
>>> bool('spam') # Object's Boolean value
>>> X = None # None placeholder
>>> print(X)
>>> L = [None] * 100 # Initialize a list of 100 Nones
>>> L
```