CPSC 1301, Computer Science I Lab Assignment

Learning Python, 5th Edition, Chapter00 Lab

Run the following commands in your Python interpreter. These are from *Learning Python, 5th Edition*, Chapter00. Submit a transcript of your session as your deliverable for this lab. This should be a plain text file named pythonlab-Chapter00_lastname.txt.

```
>>> 40 + 3.14 # Integer to float, float math/result
>>> int(3.1415) # Truncates float to integer
>>> float(3) # Converts integer to float
>>> a = 3 # Name created: not declared ahead of time
>>> b = 4
>>> a + 1, a - 1 # Addition (3 + 1), subtraction (3 - 1)
>>> b * 3, b / 2 # Multiplication (4 * 3), division (4 / 2)
>>> a % 2, b ** 2 # Modulus (remainder), power (4 ** 2)
>>> 2 + 4.0, 2.0 ** b # Mixed-type conversions
>>> c * 2
>>> b / 2 + a # Same as ((4 / 2) + 3) [use 2.0 in 2.X]
>>> b / (2.0 + a) # Same as (4 / (2.0 + 3)) [use print before 2.7]
>>> b / (2.0 + a) # Pythons <= 2.6: echoes give more (or fewer) digits
>>> print(b / (2.0 + a)) # But print rounds off digits
>>> 1 / 2.0
>>> num = 1 / 3.0
>>> num # Auto-echoes
>>> print(num) # Print explicitly
>>> '%e' % num # String formatting expression
>>> '%4.2f' % num # Alternative floating-point format
>>> '{0:4.2f}'.format(num) # String formatting method: Python 2.6, 3.0, and later
>>> repr('spam') # Used by echoes: as-code form
>>> str('spam') # Used by print: user-friendly form
>>> 1 < 2 # Less than
>>> 2.0 >= 1 # Greater than or equal: mixed-type 1 converted to 1.0
>>> 2.0 == 2.0 # Equal value
>>> 2.0 != 2.0 # Not equal value
>>> X = 2
>>> Y = 4
>>> Z = 6
>>> X < Y < Z # Chained comparisons: range tests
>>> X < Y and Y < Z
>>> X < Y > Z
>>> X < Y and Y > Z
False>>> import math
>>> 5 / -2 # Keep remainder
>>> 5 // -2 # Floor below result
>>> math.trunc(5 / -2) # Truncate instead of floor (same as int())
>>> 1 < 2 < 3.0 < 4
>>> 1 > 2 > 3.0 > 4
>>> 1 == 2 < 3 # Same as: 1 == 2 and 2 < 3
```

```
>>> 1.1 + 2.2 == 3.3 # Shouldn't this be True?...
>>> 1.1 + 2.2 # Close to 3.3, but not exactly: limited precision>>> 5 // 2.0, 5 // -2.0 # Ditto for floa
>>> int(1.1 + 2.2) == int(3.3) # OK if convert: see also round, floor, trunc ahead
>>> 10 / 4 # Differs in 3.X: keeps remainder
>>> 10 / 4.0 # Same in 3.X: keeps remainder
>>> 10 // 4 # Same in 3.X: truncates remainder
>>> 10 // 4.0 # Same in 3.X: truncates to floor
>>> import math
>>> math.floor(2.5) # Closest number below value
>>> math.floor(-2.5)
>>> math.trunc(2.5) # Truncate fractional part (toward zero)
>>> math.trunc(-2.5)
>>> 5 / 2, 5 / -2
>>> 5 // 2, 5 // -2 # Truncates to floor: rounds to first lower integer
>>> 5 / 2.0, 5 / -2.0
>>> import math
>>> 5 / -2 # Keep remainder
>>> 5 // -2 # Floor below result
>>> math.trunc(5 / -2) # Truncate instead of floor (same as int())
>>> (5 / 2), (5 / 2.0), (5 / -2.0), (5 / -2) # 3.X true division
>>> (5 // 2), (5 // 2.0), (5 // -2.0), (5 // -2) # 3.X floor division
>>> (9 / 3), (9.0 / 3), (9 // 3), (9 // 3.0) # Both
>>> 2 ** 200
>>> x = 1 # 1 decimal is 0001 in bits
>>> x << 2
>>> x | 2 # Bitwise OR (either bit=1): 0011
>>> x & 1 # Bitwise AND (both bits=1): 0001
>>> import math
>>> math.pi, math.e # Common constants
>>> math.sin(2 * math.pi / 180) # Sine, tangent, cosine
>>> math.sqrt(144), math.sqrt(2) # Square root
>>> pow(2, 4), 2 ** 4, 2.0 ** 4.0 # Exponentiation (power)
>>> abs(-42.0), sum((1, 2, 3, 4)) # Absolute value, summation
>>> min(3, 1, 2, 4), max(3, 1, 2, 4) # Minimum, maximum
>>> math.floor(2.567), math.floor(-2.567) # Floor (next-lower integer)
>>> math.trunc(2.567), math.trunc(-2.567) # Truncate (drop decimal digits)
>>> int(2.567), int(-2.567) # Truncate (integer conversion)
>>> round(2.567), round(2.467), round(2.567, 2) # Round (Python 3.X version)
>>> '%.1f' % 2.567, '{0:.2f}'.format(2.567) # Round for display (Chapter 7)
>>> (1 / 3.0), round(1 / 3.0, 2), ('%.2f' % (1 / 3.0))
>>> import math
>>> math.sqrt(144) # Module
>>> 144 ** .5 # Expression
>>> pow(144, .5) # Built-in
>>> math.sqrt(1234567890) # Larger numbers
>>> 1234567890 ** .5
>>> pow(1234567890, .5)
>>> import random
>>> random.random()
>>> random.random() # Random floats, integers, choices, shuffles
>>> random.randint(1, 10)
>>> random.randint(1, 10)
>>> random.choice(['Life of Brian', 'Holy Grail', 'Meaning of Life'])
>>> random.choice(['Life of Brian', 'Holy Grail', 'Meaning of Life'])
```

```
>>> suits = ['hearts', 'clubs', 'diamonds', 'spades']
>>> random.shuffle(suits)
>>> suits
>>> random.shuffle(suits)
>>> suits
>>> from fractions import Fraction
>>> x = Fraction(1, 3) # Numerator, denominator
>>> y = Fraction(4, 6) # Simplified to 2, 3 by gcd
>>> x
>>> y
>>> print(y)
>>> x + y
>>> x - y # Results are exact: numerator, denominator
>>> x * y
>>> Fraction('.25')
>>> Fraction('1.25')
>>>
>>> Fraction('.25') + Fraction('1.25')
>>> a = 1 / 3.0 # Only as accurate as floating-point hardware
>>> b = 4 / 6.0 # Can lose precision over many calculations
>>> a
>>> b
>>> a + b
>>> a - b
>>> a * b
>>> x = set('abcde')
>>> y = set('bdxyz')
>>> x
>>> x - y # Difference
>>> x | y # Union
>>> x & y # Intersection
>>> x ^ y # Symmetric difference (XOR)
>>> x > y, x < y # Superset, subset
>>> 'e' in x # Membership (sets)
>>> 'e' in 'Camelot', 22 in [11, 22, 33] # But works on other types too
>>> z = x.intersection(y) # Same as x & y
>>> z
>>> z.add('SPAM') # Insert one item
>>> z
>>> z.update(set(['X', 'Y'])) # Merge: in-place union
>>> z
>>> z.remove('b') # Delete one item
>>> z
>>> for item in set('abc'): print(item * 3)
>>> S | set([3, 4]) # Expressions require both to be sets
>>> S | [3, 4]
>>> S.union([3, 4]) # But their methods allow any iterable
>>> S.intersection((1, 3, 5))
>>> S.issubset(range(-5, 5))
>>> set([1, 2, 3, 4]) # Built-in: same as in 2.6
>>> set('spam') # Add all items in an iterable
>>> {1, 2, 3, 4} # Set literals: new in 3.X (and 2.7)
>>> S = {'s', 'p', 'a', 'm'}
>>> S
```

```
>>> S.add('alot') # Methods work as before
>>> S
>>> S1 = {1, 2, 3, 4}
>>> S1 & {1, 3} # Intersection
>>> {1, 5, 3, 6} | S1 # Union
>>> S1 - {1, 3, 4} # Difference
>>> S1 > {1, 3} # Superset
>>> S1 - {1, 2, 3, 4} # Empty sets print differently
>>> type({}) # Because {} is an empty dictionary
>>> S = set() # Initialize an empty set
>>> S.add(1.23)
>>> S
>>> S1 - {1, 2, 3, 4} # Empty sets print differently
>>> type({}) # Because {} is an empty dictionary
>>> S = set() # Initialize an empty set
>>> S.add(1.23)
>>> S
>>> S
>>> S.add([1, 2, 3]) # Only immutable objects work in a set
>>> S.add({'a':1})
>>> S.add((1, 2, 3))
>>> S # No list or dict, but tuple OK
>>> S | {(4, 5, 6), (1, 2, 3)} # Union: same as S.union(...)
>>> (1, 2, 3) in S # Membership: by complete values
>>> (1, 4, 3) in S
>>> {x ** 2 for x in [1, 2, 3, 4]} # 3.X/2.7 set comprehension
>>> {x for x in 'spam'} # Same as: set('spam')
>>> {c * 4 for c in 'spam'} # Set of collected expression results
>>> {c * 4 for c in 'spamham'}
>>> S = {c * 4 for c in 'spam'}
>>> S | {'mmmm', 'xxxx'}
>>> S & {'mmmm', 'xxxx'}
>>> L = [1, 2, 1, 3, 2, 4, 5]
>>> set(L)
>>> L = list(set(L)) # Remove duplicates
>>> L
>>> list(set(['yy', 'cc', 'aa', 'xx', 'dd', 'aa'])) # But order may change
>>> set([1, 3, 5, 7]) - set([1, 2, 4, 5, 6]) # Find list differences
>>> set('abcdefg') - set('abdghij') # Find string differences
>>> set('spam') - set(['h', 'a', 'm']) # Find differences, mixed
>>> set(dir(bytes)) - set(dir(bytearray)) # In bytes but not bytearray
>>> set(dir(bytearray)) - set(dir(bytes))
>>> L1, L2 = [1, 3, 5, 2, 4], [2, 5, 3, 4, 1]
>>> L1 == L2 # Order matters in sequences
>>> set(L1) == set(L2) # Order-neutral equality
>>> sorted(L1) == sorted(L2) # Similar but results ordered
>>> 'spam' == 'asmp', set('spam') == set('asmp'), sorted('spam') == sorted('asmp')
>>> engineers = {'bob', 'sue', 'ann', 'vic'}
>>> managers = {'tom', 'sue'}
>>> 'bob' in engineers # Is bob an engineer?
>>> engineers & managers # Who is both engineer and manager?
>>> engineers | managers # All people in either category
>>> engineers - managers # Engineers who are not managers
>>> managers - engineers # Managers who are not engineers
```

```
>>> engineers > managers # Are all managers engineers? (superset)
>>> {'bob', 'sue'} < engineers # Are both engineers? (subset)
>>> (managers | engineers) > managers # All people is a superset of managers
>>> managers ^ engineers # Who is in one but not both?
>>> (managers | engineers) - (managers ^ engineers) # Intersection!
>>> type(True)
>>> isinstance(True, int)
>>> True == 1 # Same value
>>> True is 1 # But a different object: see the next chapter
>>> True or False # Same as: 1 or 0
>>> True + 4 # (Hmmm)
```