CPSC 1301, Computer Science I Lab Assignment

Learning Python, 5th Edition, Chapter10 Lab

Run the following commands in your Python interpreter. These are from *Learning Python, 5th Edition*, Chapter10. Submit a transcript of your session as your deliverable for this lab. This should be a plain text file named pythonlab-Chapter10_lastname.txt.

1 Chapter 10

```
>>> reply = '20'
>>> reply ** 2
>>> S = '123'
>>> T = 'xxx'
>>> S.isdigit(), T.isdigit()


while True:
    reply = input('Enter text:')
    if reply == 'stop': break
    print(reply.upper())
print('Bye')


while True:
    reply = input('Enter text:')
        if reply == 'stop':
        break
    elif not reply.isdigit():
        print('Bad!' * 8)
    else:
        print(int(reply) ** 2)
print('Bye')


while True:
    reply = input('Enter text:')
    if reply == 'stop':
        break
    try:
        num = int(reply)
    except:
        print('Bad!' * 8)
    else:
        print(num ** 2)
print('Bye')


while True:
    reply = input('Enter text:')
    if reply == 'stop':
        break
    try:
        print(float(reply) ** 2)
    except:
        print('Bad!' * 8)
print('Bye')
```

```
while True:
    reply = input('Enter text:')
    if reply == 'stop':
        break
    elif not reply.isdigit():
        print('Bad!' * 8)
    else:
        num = int(reply)
        if num < 20:
            print('low')
        else:
            print(num ** 2)
print('Bye')
```

2 Chapter 11

```
>>> nudge = 1 # Basic assignment
>>> wink = 2
>>> A, B = nudge, wink # Tuple assignment
>>> A, B # Like A = nudge; B = wink
>>> [C, D] = [nudge, wink] # List assignment
>>> C, D
>>> nudge = 1
>>> wink = 2
>>> nudge, wink = wink, nudge # Tuples: swaps values
>>> nudge, wink # Like T = nudge; nudge = wink; wink = T
>>> [a, b, c] = (1, 2, 3) # Assign tuple of values to list of names
>>> a, c
>>> (a, b, c) = "ABC" # Assign string of characters to tuple
>>> a, c
>>> string = 'SPAM'
>>> a, b, c, d = string # Same number on both sides
>>> a, d
>>> a, b, c = string # Error if not
>>> a, b, c = string[0], string[1], string[2:] # Index and slice
>>> a, b, c
>>> a, b, c = list(string[:2]) + [string[2:]] # Slice and concatenate
>>> a, b, c
>>> a, b = string[:2] # Same, but simpler
>>> c = string[2:]
>>> a, b, c
>>> (a, b), c = string[:2], string[2:] # Nested sequences
>>> a, b, c
>>> ((a, b), c) = ('SP', 'AM') # Paired by shape and position
>>> a, b, c
>>> red, green, blue = range(3)
>>> red, blue
>>> list(range(3)) # list() required in Python 3.X only
>>> L = [1, 2, 3, 4]
>>> while L:
>>>     front, L = L[0], L[1:] # See next section for 3.X * alternative
>>>     print(front, L)
>>>
>>> seq = [1, 2, 3, 4]
>>> a, b, c, d = seq
>>> print(a, b, c, d)
>>> a, b = seq
>>> a, *b = seq
>>> a
>>> b
>>> *a, b = seq
>>> a
>>> b
>>> a, *b, c = seq
>>> a
>>> b
>>> c
>>> a, b, *c = seq
>>> a
>>> b
>>> c
>>> a, *b = 'spam'
>>> a, b
>>> a, *b, c = 'spam'
>>> a, b, c
>>> a, *b, c = range(4)
>>> a, b, c
>>> S = 'spam'
>>> S[0], S[1:] # Slices are type-specific, * assignment always returns a list
>>> S[0], S[1:3], S[3]
>>> L = [1, 2, 3, 4]
>>> while L:
>>>     front, *L = L # Get first, rest without slicing
>>>     print(front, L)
>>>
>>> seq = [1, 2, 3, 4]
>>> a, b, c, *d = seq
>>> print(a, b, c, d)
>>> a, b, c, d, *e = seq
>>> print(a, b, c, d, e)
>>> a, b, *e, c, d = seq
>>> print(a, b, c, d, e)
>>> a, *b, c, *d = seq
>>> a, b = seq
>>> *a = seq
>>> *a, = seq
>>> a
>>> seq
>>> a, *b = seq # First, rest
>>> a, b
>>> a, b = seq[0], seq[1:] # First, rest: traditional
>>> a, b
>>> *a, b = seq # Rest, last
>>> a, b
>>> a, b = seq[:-1], seq[-1] # Rest, last: traditional
>>> a, b
>>> *a, b = seq # Rest, last
>>> a, b
>>> a, b = seq[:-1], seq[-1] # Rest, last: traditional
>>> a, b
>>> a = b = 0
>>> b = b + 1
>>> a, b
>>> a = b = []
>>> b.append(42)
>>> a, b
>>> a = []
>>> b = [] # a and b do not share the same object
>>> b.append(42)
>>> a, b
>>> a, b = [], [] # a and b do not share the same object
>>> x = 1
>>> x = x + 1 # Traditional
>>> x
>>> x += 1 # Augmented
>>> x
>>> S = "spam"
>>> S += "SPAM" # Implied concatenation
>>> S
>>> L = [1, 2]
>>> L = L + [3] # Concatenate: slower
>>> L
>>> L.append(4) # Faster, but in place
>>> L
>>> L = L + [5, 6] # Concatenate: slower
>>> L
>>> L.extend([7, 8]) # Faster, but in place
>>> L
>>> L += [9, 10] # Mapped to L.extend([9, 10])
>>> L
>>> L = []
>>> L += 'spam'
>>> L = L + 'spam'
>>> L = [1, 2]
>>> M = L # L and M reference the same object
>>> L = L + [3, 4] # Concatenation makes a new object
>>> L, M # Changes L but not M
>>> L = [1, 2]
>>> M = L
>>> L += [3, 4] # But += really means extend
>>> L, M # M sees the in-place change too!
>>> L = [1, 2]
>>> L.append(3) # Append is an in-place change
>>> L
>>> L = L.append(4) # But append returns None, not L
>>> print(L) # So we lose our list!
>>> print() # Display a blank line
>>> x = 'spam'
>>> y = 99
>>> z = ['eggs']
>>>
>>> print(x, y, z) # Print three objects per defaults
>>> print(x, y, z, sep='') # Suppress separator
>>>
>>> print(x, y, z, sep=', ') # Custom separator
>>> print(x, y, z, end='') # Suppress line break
>>>
>>> print(x, y, z, end=''); print(x, y, z) # Two prints, same output line
>>> print(x, y, z, end='...\n') # Custom line end
>>> print(x, y, z, sep='...', end='!\n') # Multiple keywords
>>> print(x, y, z, end='!\n', sep='...') # Order doesn't matter
>>> print(x, y, z, sep='...', file=open('data.txt', 'w')) # Print to a file
>>> print(x, y, z) # Back to stdout
>>> print(open('data.txt').read()) # Display file text
```