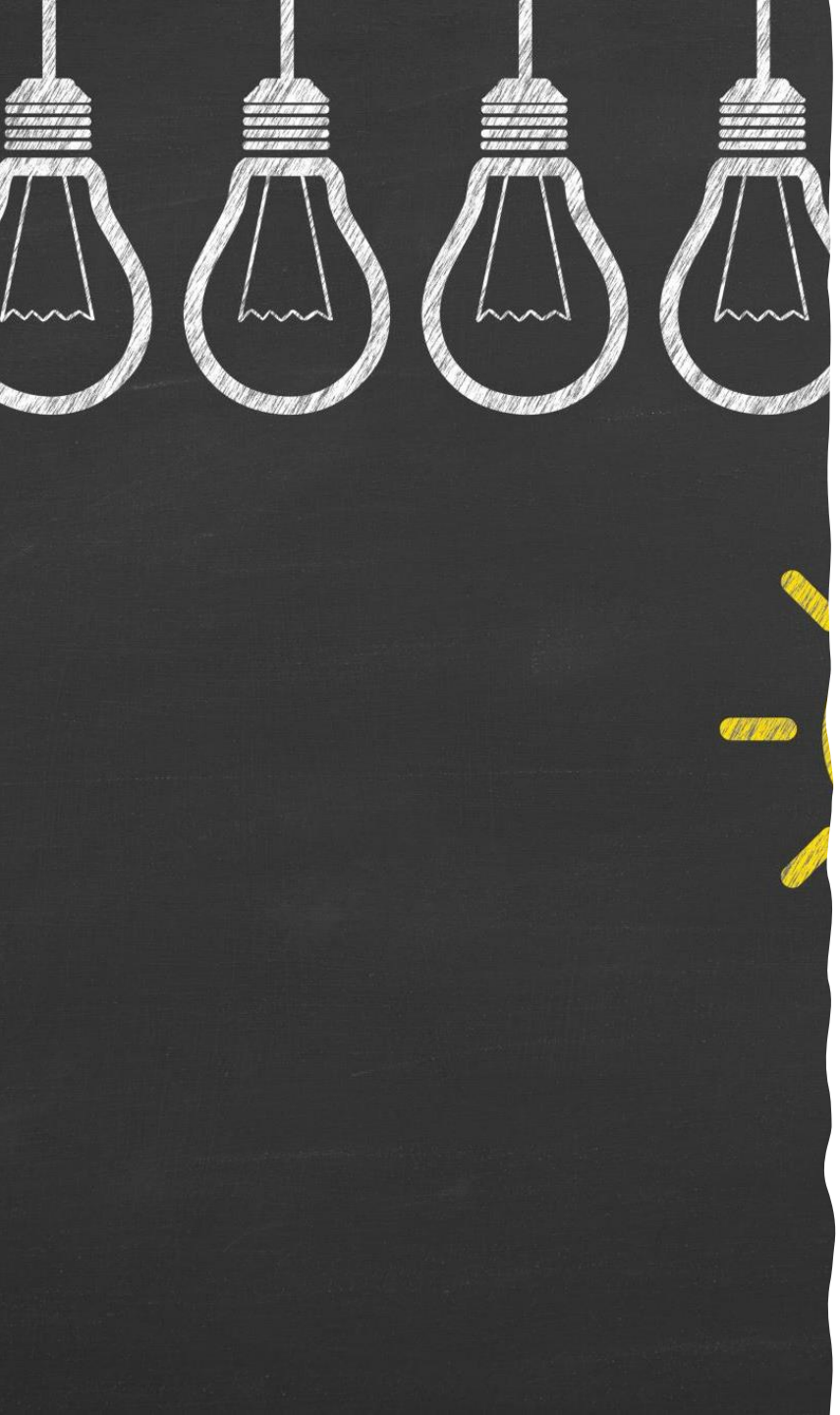


CMSC 105
Elementary Programming

Acknowledgement: These slides are adapted from slides provided with “Introduction to Programming Using Python, Liang (Pearson 2013)” and slides shared by Dr. Jory Denny

Outline

Lists

Practice Exercises

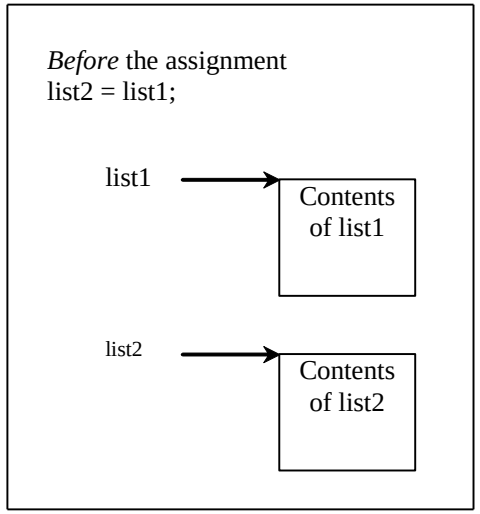
Copying Lists

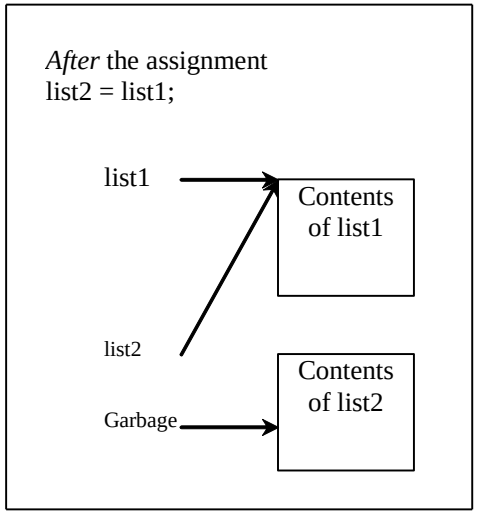
- Often, in a program, you need to duplicate a list or a part of a list.
- In such cases, you could attempt to use the assignment statement (=), as follows:

```
l2 = l1
```

- But this copies the reference, not the list. Do this trick instead:

```
l2 =  [] + l1
  # Append to a new
list
```





Heterogeneous lists

A list can contain values of different types.

```
>>> s = [‘Score’, 80]
>>> y = [‘Year’, 2015]
>>> result = [s,y]
>>> result
[[‘Score’, 80],['Year', 2015]]
```

String and a number

List of lists, or nested list

Passing lists to functions

- Passing lists to functions is perfectly normal. Consider:

```
def printList(lst):
    for x in lst:
        print(x, end=" ")
    print()

l = [3, 1, 2, 6, 4, 2]
printList(l)

printList(["Hi", 5, 2.3]) # Anonymous list
```

Mutable vs Immutable Objects

- An immutable object can’t be changed after it is created.

  Example: int, float, bool, string.

- Mutable objects are easy to change.

  Example: list, dictionary, set.

Passing lists to functions

- Python uses **pass-by-object-reference** to pass arguments to a function. There are important differences between passing the values of variables of numbers and strings and passing lists.

  - Immutable objects act like pass-by-value (numbers and strings)
  - Mutable objects can have their memory altered (lists and other objects)

Passing lists to functions

Example

```
def main():
  x = 1           # x represents an int value
  y = [1, 2, 3] # y represents a list
  m(x, y)         # Invoke f with arguments x and y
  print("x is " + str(x))          # Prints 1, not 1001
  print("y[0] is " + str(y[0])) # Prints 5555

def m(number, numbers):
  number = 1001      # Assign a new value to number
  numbers[0] = 5555 # Assign a new value to
numbers[0]

main()
```

Subtle Issues Regarding Default Arguments

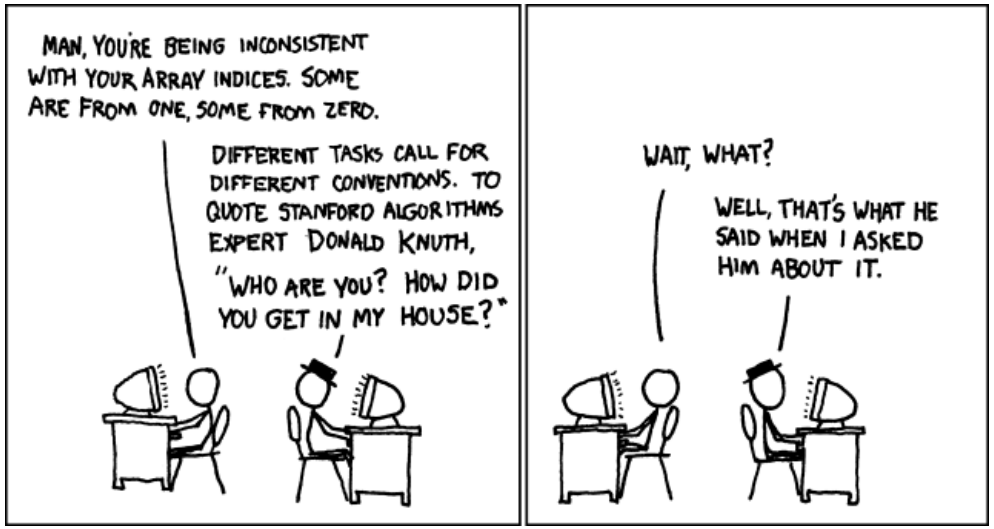
```
def add(x, lst = []):
    if x not in lst:
        lst.append(x)
    return lst

l1 = add(1)
print(l1) # [1]
l2 = add(2)
print(l2) # [1, 2]
l3 = add(3, [11, 12,
13])
print(l3) # [11, 12,
13, 3]
l4 = add(4)
print(l4) # [1, 2, 4]
```

- Default values are only created only once.
- Consider this program. Its output is:

```
[1]
[1, 2]
[11, 12, 13, 3]
[1, 2, 4]
```

Summary

- Lists.
  - Organized way to store huge quantities of data.
  - Almost as easy to use as primitive types.
  - Can directly access an element given its index.



http://imgs.xkcd.com/comics/donald_knuth.png

Problem 1

- Write a program that reads two inputs—
  - Names separated by spaces
  - Scores separated by spaces

Run a for loop for each element in names and print the corresponding scores. For example,

```
Enter names: Bob Ana John
Enter scores:78 87 98

Output:
Bob-78
Ana-87
John-98
```

Problem 2

Write a program that takes as input a sequence of numbers separated by spaces. Using list comprehension, print alternate elements of the original sequence.

Example:

```
Enter sequence:1 2 3 4 5
Output:
1 3 5
```

Problem 3

Show the output of the following:

```
def m(x,y):
    x=3
    y[0]=3

def main():
    number=0
    numbers=[10]
    m(number,numbers)
    print("Number is", number,"and numbers[0]
is",numbers[0])
main()
```

Problem 4

What will be the output?

```
def modify_list(num,lst=[1,2,3]):
    lst.append(num)
    return lst
print(modify_list(12))
print(modify_list(1,[11,12,13]))
print(modify_list(14))
print(modify_list(1,[2,22,23]))
```

Chapters Covered from Textbook

- Chapter 10

Thank you!
Questions?